1967

The use of interrupts in a time-sharing computer system.

Egerton, James White

Monterey, California. U.S. Naval Postgraduate School

THE USE OF INTERRUPTS IN A TIME-SHARING COMPUTER SYSTEM

JAMES WHITE EGERTON

THE USE OF INTERRUPTS

IN A TIME-SHARING COMPUTER SYSTEM

by

James White Egerton
Lieutenant Commander, United States Navy
B.S., Naval Academy, 1956

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMMUNICATIONS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1967

ABSTRACT

Digital computer interrupts are becoming more important as these machines increase in the interaction with their environment. Different methods of interrupt implementation are described. They are then analyzed in the areas of response time, overhead, and saturation. Examples of the use of interrupts in different computational environments are given. Five modifications to a general purpose computer system are proposed. These modifications, each of which used interrupts, enable the system to be more easily used in a limited time-sharing mode. The results of these modifications are compared to those of the software that would be required to accomplish the same objectives.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

TABLE OF SYMBOLS AND ABBREVIATIONS

Symbol	Meaning
A	Twenty-four bit accumulator register
AO-A23	Stages of the accumulator
A/D	Analog to digital
Add1-3	Three stages of the full adder
Ai	Allow interrupt flip-flop
B	Twenty-four bit accumulator extension register
BRX	Signal which is true during an Increment Index and Branch instruction
C	Twenty-four bit exchange register
CO-C23	Stages of the exchange register
cps	characters per second
CPU	Central processing unit
Cr3	Signal which shifts the exchange register right three bits at each timing pulse
CT16	Medium frequency crystal oscillator module in SDS T Series Logic
D/A	Digital to analog
DCL	Digital Control Laboratory
EAX	Signal which is true during an Effective Address into X instruction
Eax	Flip-flop set during an Effective Address into X instruction and programmed operator execution
En	Signal which enables the interrupt system
(En)	Switch which enables the interrupt system
End	Signal which is true during the last phase of an instruction
Eom	Signal which is true during an Energize Output, Multiplex instruction

F1-F3	Three-stage counter which determines the phase of an instruction cycle
Go	Flip-flop which is set during computer operation
I	Index bit (bit 9) of an instruction
Ia	Flip-flop used for indirect addressing and for advancing the program counter
Ib	Signal which indicates completion of an interrupt subroutine
ICS	Interrupt control system
Ie	Signal which indicates that the computer has started processing an interrupt subroutine
Ij	Signal which indicates that the present interrupt is a single-instruction interrupt
Inr	Interrupt interlock flip-flop
Int	Interrupt flip-flop
I/O	Input/output
ips	Inches per second
Ip5-Ip900	Interrupt process flip-flops
Ir	Interrupt signal for an enable-required interrupt
Is	Interrupt signal for an enable-not-required interrupt
Is5-Is900	Interrupt storage flip-flops
Ix	Index flip-flop
I5-- I900	Interrupt request signals
Ju	Branching flip-flop
K	One thousand
kHz	Kilo-Hertz
(Kmc)	Memory clear switch
(Kr)	Hold switch
M	Operand address of an instruction (bits 10-23)

mHz	Mega-Hertz
Mo	Mode flip-flop
Mp1 - Mp3	Three stages of the memory protect switch
msec	Millisecond
NI	Next instruction
NOP	Signal which is true during a No Operation instruction
N5-N14	Interrupt address lines
O	Six bit operation code register
Ob	Signal which indicates that a memory protect violation has occurred
Oc	Signal which sets the operation code register to 020
OPCODE	Operation code
Oxc	Signal which enables the transfer of C to O
O1-O6	Stages of the operation code register
P	Program address register; programmed operator bit (bit2) of an instruction
Pi	Flip-flop which is set upon execution user mode privileged instruction
Pid	Signal which is enabled by a privileged instruction opcode in the exchange register
Piq	Signal which is true when a privileged instruction is executed in user mode
Pme	Signal which is true when the storage address register contains a protected memory reference
POP	Programmed operator
Pr3	Signal which shifts the program address register right three bits at each clock pulse
P0-P14	Stages of the program address register
Qtc	Output signal of the quantum timer oscillator
Qt1-4	Four flip-flops of the quantum timer control circuit

Q1-Q6	Six-stage counter which determines the designation of a timing pulse
R	Relocation bit (bit 0) of an instruction
RCH	Signal which is true during a Register Change instruction
Rf	Ready flip-flop
Rtc	Signal used for real-time clock input
Rt1-3	Three flip-flops of the real-time clock control circuit
rXYZ	Reset flip-flop XYZ
S	Storage address register
SDS	Scientific Data Systems, Inc.
Sk	Skip flip-flop
Ski	Signal which initiates the CLOCK SYNC interrupt
Sku	Signal which initiates the end of time quantum interrupt
(St)	Signal enabled by the START switch
Sxc	Signal which enables the transfer of C to S
Sxn	Signal which enables the transfer of an interrupt address to S
Sxp	Signal which enables the transfer of P to S
Sx48	Signal used in conjunction with shift instructions
sXYZ	Set flip-flop XYZ
S1-S14	Stages of the storage address register
Ti	Signal which is true from T3 through T0
TMCC	Time multiplexed communication channel
Tp	Last timing pulse of a machine cycle
Tr	Next to last timing pulse of a machine cycle
Ts	Signal which is true during a time-shared memory operation

Tsm	Flip-flop used during interlaced I/O operation
Tsr	Flip-flop used during interlaced I/O operation
T8,T7,...,T0	First nine timing pulses of a machine cycle
usec	Microsecond
Usi	User mode single-instruction flip-flop
X	Index register; indexing bit (bit 1) of an instruction
Xwl	Most significant write flip-flop
Ø0-Ø7	Eight phase designations of an instruction cycle
33	Decimal number
033	Octal number
	Indicates signal originates at a switch
$\overline{XYZ}$	Indicates XYZ is false or zero

SECTION I

INTRODUCTION

Early digital computers were designed and used to solve individual problems involving extensive calculations and produce the answers. When the tremendous computational power of computers was applied to other uses, however, the mere sequential execution of programmed steps in a predetermined order was no longer sufficient. In such areas as on-line automatic control, real-time simulation, and time-sharing systems, the computer is required to be responsive to the demands of its environment. This requirement is usually fulfilled by the use of interrupts.

Basically, an interrupt is a circuit designed to halt the normal execution of a computer program on signal, and to execute in its place a group of instructions to fulfill the needs of the device which initiated the signal.

This paper will analyze the various methods of interrupt implementation and usage, and propose a small-scale time-sharing system which is implemented on a general purpose digital computer through design changes utilizing priority interrupts.

SECTION II

ANALYSIS OF INTERRUPT SYSTEM REQUIREMENTS

Interrupts can be divided into three categories according to purpose. The first of these, the request interrupt, is one which requires the computer to take some immediate action to satisfy the cause of the interrupt. Secondly, the indicator interrupt notifies the computer of some external condition which does not require action, but the computer may sample this condition to determine a course of action. The third type is the fault interrupt, which is caused by some internal abnormal condition such as a parity error.

I. DESIGN CHARACTERISTICS

There are many methods for implementing interrupts in a digital computer. The following paragraphs will describe the most prevalent methods and classify them logically according to their characteristics.

Methods of Implementation

The basic requirement for an interrupt is to stop execution of the current instruction sequence, make some record of the stopping point, and execute instead some other special set of instructions, called the interrupt subroutine. The first method to implement this is the single interrupt system.

Single interrupt. The single interrupt system basically consists of one flip-flop in the computer's control circuitry which is wired to some external device. This flip-flop is set whenever the condition causing the interrupt is present. The CPU will normally check the state of this flip-flop after each instruction execution. If it is set, the CPU will store the contents of the program address counter and execute the next instruction (the first instruction in the interrupt subroutine) from

some fixed location associated with this interrupt. Some provision would then be made to clear the flip-flop and return to the point of interruption upon completion of the subroutine. The use of a system that does not mark the point of interruption is not feasible for the following reasons. Request and indicator interrupts normally require returning to the program in progress when the interrupt was received. Fault interrupts usually would not return, but the address of the instruction being executed when the fault occurred is important in determining the cause of the fault.

The single interrupt scheme described above has another limitation in addition to being able to sense only one external condition. If, while the interrupt is being processed, a second interrupt signal is received, it will be lost, since it will attempt to set a flip-flop which is already set. If the flip-flop is cleared at the start of the interrupt subroutine rather than at the end to correct this, then another interrupt signal would start the subroutine over again, destroying the original interruption point. This would result in an infinite program loop.

Single interrupt with interlock. The deficiency noted above can be eliminated by using a single interrupt with interlock, a device similar to the single interrupt except that it has a second, or interlock, flip-flop. An interrupt signal sets the first, or interrupt, flip-flop. At the end of the next instruction, the CPU initiates the interrupt only if the interlock flip-flop is cleared. Immediately after starting the interrupt subroutine, the CPU sets the interlock flip-flop and clears the interrupt flip-flop. Another interrupt signal may then be received and stored until completion of the present interrupt subroutine, when the interlock flip-flop is cleared.

Multiple interrupt with interlock. The next logical step in interrupt system development is a system which will accept interrupt signals from more than one source and be able to determine which source originated the signal. Such a multiple interrupt system operates in a manner similar to the single interrupt, except that the interrupt signal, in addition to setting the interrupt flip-flop, also sets some indicator in the computer which is unique to the signal source. Once the interrupt subroutine has been entered, it interrogates the source indicators to determine which external device initiated the signal.

When this multiple source concept is applied to the single interrupt, new problems must be resolved. Since multiple interrupts are usually independent, the probability exists that an interrupt from one source may still be in process when an interrupt from another source is received. The system should certainly prevent this second signal from being lost. As a minimum requirement, therefore, an interlock should be used with a multiple interrupt system.

The multiple interrupt with interlock can store a second interrupt signal while processing the first one. This system must clear the source indicator immediately after interrogating it. Otherwise, when the second signal sets its indicator, an ambiguity would exist as to which source initiated the second interrupt. When this idea is extended to allow for more than one additional interrupt, the interrupt flip-flop then becomes an indication that at least one interrupt signal was received while processing the present interrupt. The CPU would interrogate and store all source indicators for orderly processing. The system would then be returned to the condition of being receptive to interrupts from all sources. If this interrupt scheme were used, the system would have to be designed

so that interrupts from one source occurring more often than they could be serviced would either be impossible or of no significance.

Another implication of the interlock flip-flop is that while an interrupt is being processed, the computer is uninterruptible. This is always desirable for the single interrupt for reasons previously mentioned. It is possible, however, that in a multiple interrupt system, an interrupt signal is received which is more important than the one being processed. In such a situation it may be desirable to allow the more important interrupt to gain control in deference to the less important one. A system with this feature is one form of a priority interrupt system.

Priority interrupt. A priority interrupt system is basically defined as an interrupt system which, when two or more interrupt signals are involved, is able to distinguish among them according to priority and process the one with higher priority. The first of two types of priority interrupt systems to be discussed is the single level priority interrupt system. This system has essentially the same features as the multiple interrupt with interlock, with the added condition that when the CPU finds the interrupt flip-flop set, it interrogates the source indicators one at a time, in order of priority, to determine the next interrupt to be processed. It is single level because the CPU is uninterruptible while executing an interrupt subroutine.

The second type is the multi-level priority interrupt system. This system is similar to the one previously described, with the exception that an interrupt subroutine may be interrupted by a higher priority interrupt signal. This system has the advantage that if an interrupt subroutine is being processed, it will always be the most important of the ones awaiting processing.

In some situations where priority interrupts are used, a special

interrupt is necessary for processing computer faults. This special device, known as a trap, is similar to an interrupt except that control is not returned to the interrupted program, which eliminates the problem of register and indicator storage. A trap usually does not require a position in the priority hierarchy, since the normal purpose of a trap routine is to make a post-mortem examination of the faulted program, a task of inherently low priority.

Selectivity. Another important characteristic of interrupt systems is selectivity. Selectivity is the ability of the computer to ignore or defer action on all or selected individual interrupts. Many ambiguous terms related to selectivity are used in the computer field. The terminology used here has been chosen as the most logically descriptive in each case.

The first selectivity feature to be discussed is recognize/ignore. The computer may, under program control, select either the recognize state or the ignore state for an interrupt. If the ignore state is selected, all incoming interrupt signals are essentially grounded. Recognize/ignore may be further classified as general or selective. The general recognize/ignore pertains to the entire interrupt system. The selective recognize/ignore has the ability to ignore only part of the system and recognize interrupts from the remainder. Selective recognize/ignore is classed as either individual, for single interrupts, or group, where more than one is involved. If this feature is designed so that a general recognize following a general ignore will return the selective ignores to their original state, then the selective recognize/ignore is said to be exclusive. Conversely, an inclusive selective recognize/ignore would be one in which previously executed individual or group ignores are destroyed by a general ignore.

That feature which permits the computer to postpone incoming interrupt signals for later process is called allow/defer. When one or more interrupts are deferred (placed in the defer state), interrupt signals are stored in the source indicators. An interrupt signal is processed normally when that interrupt is allowed (returned to the allow state). The characteristics of selectivity for the recognize/ignore feature are equally applicable to allow/defer.

Return of control. An important consideration in interrupt implementation is the method of retaining the status of the interrupted program so that a smooth transition can be made back to it when the interrupt subroutine is completed. The minimum requirement, as stated previously, is to save the program address counter. It is desirable, however, to provide some hardware method of storing any internal computer indicators, such as overflow. When the computer is to return to a program after an interrupt subroutine, the previous contents of arithmetic and index registers must be available. The practice of saving and restoring registers when writing normal subroutines is usually an adequate method for interrupt subroutines. Of course, only the registers used need to be saved and restored.

Interruptibility. Program interruptibility is another important consideration in designing computers which use interrupts. The basic question is one of determining at what points in the execution of instructions an interrupt will be allowed to occur. The simplest logical solution to this question is to allow the recognition of interrupts between instructions. Any general scheme of allowing interrupts during the execution of an instruction can become exceedingly complex. An instruction usually causes the control circuitry to take certain actions con-

currently and to use results obtained early in an instruction to determine some course of action later in that same instruction. For this reason it would not be feasible to recommence in the middle of an instruction if that is where the interrupt occurred. The case in which an instruction uses registers for both computation and storing the answer upon completion is more critical. For example, if a divide instruction were interrupted when the divide operation were half completed, the contents of the accumulator would be meaningless. This type of instruction could not be repeated after interruption because at the beginning of the instruction the accumulator contains information to be used during execution. It is possible to restart such an instruction at the point of interruption, but all registers and indicators would have to be restored. This would be a complex and time-consuming operation for an arithmetic or shifting instruction; the effort is probably not worthwhile.

Some instructions take so long to execute, however, that to adopt the policy that all instructions will be allowed to proceed to completion before interruption is probably undesirable. Also, some computer operations are accomplished by a contiguous chain of instructions which would be difficult to reconstitute if interrupted.

II. OPERATING CHARACTERISTICS

The previous material represents an attempt to logically explore and categorize the possible methods of implementation and attributes of interrupt systems. New definitions have been provided where confusion heretofore existed. The following paragraphs discuss three important factors concerning interrupts: response time, overhead, and saturation. A careful analysis of these factors from the standpoint of system requirements should preclude the choice of the appropriate interrupt system in every case.

Response Time

Response time is a term which is broadly defined as the time between receipt by the computer of an interrupt signal and some specific action taken by the computer in response to that action. Three other terms have been chosen which define more precisely the concept of response time. These are reaction time, recognition time, and decision time.

Reaction time is defined to be the length of time between interrupt signal receipt and the start of execution of the first instruction related to the interrupt. Reaction time, then, is a measure of how fast the computer shifts from its current program to process the interrupt once it is received. Recognition time is defined as the length of the time between the execution of the first instruction related to the interrupt and the determination by the computer of the source (and purpose) of the interrupt. Decision time is defined as the length of time between determination of interrupt source and completion of the action required by the interrupt.

Reaction time. Required reaction time is a primary consideration in determining the degree of interruptibility and the method of implemen-

tation. If reaction time of an interrupt is considered in the absence of all other possible interrupts, then it is determined only by interruptibility. Once the possibility of other interrupts is introduced into the problem, the mean response time will increase in proportion to the frequency of interrupts and to the length of their interrupt subroutines. An interrupt in a single level priority system will have a decidedly faster reaction time in a dense interrupt environment if the priority is significantly above average. The reaction time in a multi-level priority system depends very heavily on priority. The reaction time of the highest priority interrupt is a function of interruptibility only. For those of lower priority, the reaction time will increase only in proportion to the frequency of occurrence and subroutine length of those interrupts of higher priority.

In systems where interrupts may occur only between instructions, mean reaction time due to interruptibility is approximately equal to one half of the instruction execution time. An increase or decrease in frequency of points of interruption will change reaction time accordingly.

Reaction time includes the time that an interrupt is in the defer state, since the computer can react to the interrupt only in the allow state. If an interrupt is in the ignore state, reaction time is undefined, since there is no reaction to ignored interrupts.

Recognition time. Recognition time is significant only in systems which determine the source of interrupt signals by software. As an example, a multiple interrupt with interlock system might be designed so that the source indicators are interrogated sequentially by a series of instructions until the correct source is found. The recognition time would be the time required to execute these instructions. If the example system were single level priority, then the recognition time would vary

inversely with priority. Determination of interrupt source by hardware is a feasible method of reducing response time since this method eliminates recognition time.

Decision time. Decision time is finite for all interrupts, but its length depends entirely on what action is required to process the interrupt. Decision time can be reduced in some cases by processing the critical portion of an interrupt at the time of occurrence, and deferring any less important parts for processing at a later time. These less important parts may be deferred for processing by a control program, or by a lower priority, self-initiated interrupt (where such capability is available). Decision time should be held to a minimum since it affects reaction time of lower priority interrupts.

Overhead

Overhead is defined as the difference between the time required to execute all instructions concerned with an interrupt and the time required to execute those which actually process the interrupt request. Examples of overhead are the instructions used to determine the source of an interrupt, and those which are used to establish recognize/ignore and allow/defer conditions. Overhead is obviously closely related to response time. However, under certain conditions when many interrupts are used, overhead could be high enough to markedly degrade computer performance while response time remained at a satisfactory level. Steps taken to reduce response time by accomplishing interrupt housekeeping with hardware rather than software will also reduce overhead.

Instructions used to set up selectivity conditions increase overhead without affecting response time, since these are normally executed by a control program rather than by an interrupt subroutine. The use of an

exclusive selectivity system will materially reduce overhead, but at the expense of more complex hardware.

The time required to save and restore registers and indicators is also overhead time. This can be reduced by storing only those registers which are used by the interrupt subroutine. Some third generation computers use multiple register-blocks for rapid context switching. Where this capability exists interrupts can switch to a different register block and thereby eliminate the normal save-and-restore overhead.

Saturation

Saturation is defined as the condition where information which normally enters the computer by or as the result of interrupts is lost because of excessive response time. A simple example is the case in which the time between successive interrupt signals from one source is greater than the required response time for that signal. As was stated previously, however, this condition should not normally exist, since interrupt signals from one source usually follow some cyclic pattern rather than being random. When saturation results from multiple interrupt sources, the general remedy is to use all possible methods to reduce response time and overhead. In some cases saturation can be eliminated by a change in priority. For example, if real-time clock pulses which are low in priority are delayed so much that some are lost, the priority probably could be raised enough to prevent saturation. Since the amount of processing required for clock pulses is usually small, this would have no appreciable effect on the response time of other interrupts.

The above example also illustrates an important concept, namely, that the priority assigned to an interrupt does not depend solely on its relative importance. Assigned priority should be a function of relative importance, frequency of occurrence, processing time, and type of interrupt.

SECTION III

USE OF INTERRUPTS IN COMPUTER SYSTEMS

Interrupt usage increases in proportion to the interaction between the computer and the outside world. This section will give examples of interrupt system utilization in increasing degrees of interaction.

Input/Output Interrupts

All computer systems must interact with their environment to some degree. The minimum interaction occurs in the so-called "batch processing" system, in which the computer reads in the program and data through some peripheral device, solves the problem, and transmits the answers to another peripheral device. Interrupts were first used to increase the efficiency of these input/output, or I/O, operations. Most peripheral devices operate at transmission rates much below that of the computer. An interrupt system allows the computer to execute instructions at its normal rate during and I/O operation. For example, a typical second-generation computer reads paper tape at 400 frames per second. This represents an input rate of 100 24-bit words per second, or one word each 10 msec. The computer, on the other hand, can accept and store input data at a rate of approximately 200,000 words per second, or one word every five usec. When the CPU is dedicated solely to reading paper tape, it operates at 0.05% of its maximum rate. The common interrupt scheme described below can eliminate this problem.

The paper tape reader initiates an interrupt signal to the computer each time that a word is ready to be read. The CPU could execute some program during the input operation which is not related to the information on paper tape. Receipt of the interrupt signal initiates an interrupt subroutine to store the word in memory, update the operand address

in preparation for the next word, test for end conditions, and return to the interrupted program. This operation would take approximately 20 usec, allowing the CPU to execute the unrelated program at 99.8% of its maximum rate.

The interrupt subroutine execution time can be reduced further by having a second interrupt signal initiated by the reader when an end-of-message signal is read from paper tape. If a priority interrupt system is used, this interrupt will have priority over the word-ready interrupt, otherwise the end-of-message signal would be read as data. In a multi-level priority system this second interrupt saves one instruction for each word read, since end conditions need not be checked. This instruction would also be eliminated in a single level system, but another instruction must be added to the interrupt subroutine to determine the source of the incoming interrupt signal.

Interrupts are used for all I/O devices to increase efficiency. When these devices communicate with the computer over separate communications channels, separate interrupts must be used for each channel. This causes a significant increase in subroutine execution time for the single-level priority system, since the recognition time increases in proportion to the number of system interrupts. However, such considerations are significant only from the standpoint of CPU efficiency since response time is always adequate and saturation non-existent for I/O interrupts.

On-line Real-time Interrupts

Interaction with the environment is increased significantly in on-line real-time systems. Such systems are used in automatic process control, hybrid computation and simulation, and other more specialized areas. An on-line real-time computer is defined as one which processes

data in synchronism with some real-time physical process such that the computation results are useful to the process.[1] These systems normally transmit data to and from the associated process through A/D and D/A converters.

A typical application of this system is in hybrid simulation, wherein the digital computer is interfaced with an analog computer. The digital computer is used for calculations that require a high degree of precision and for other operations more suited to this method of computation. Analog and digital computers are inherently incompatible, since the computing time in an analog computer is proportional to physical time, whereas computing time in a digital computer depends entirely on the speed of the computer and the size of the problem. The normal method of synchronizing them is to key the digital operation to periodic timing signals furnished by the analog computer, or by some external timer which furnishes these signals to both computers. The digital computer for the system must have a minimum capability of completing all required calculations initiated by a timing signal before the next signal is received.[2]

Interrupts are universally used to provide synchronization of the digital computer in a hybrid system. In the simplest system, the timing signal interrupts the computer to start a calculation cycle. When completed, the computer goes into an idle loop to await the next interrupt. Normally the digital computer will be required to perform several tasks of various lengths and frequency of occurrence. There are two methods of approaching this problem. The first places control in a digital computer executive program. The computer is interrupted periodically by timing signals. The interrupt subroutine records each timing pulse in a counter and returns control to the executive. The executive program

then initiates all tasks to be performed at the appropriate time. The second method is an expansion of the one-task system, where each task is initiated by a separate interrupt. Use of this second method is mandatory for a system in which the starting time of each digital computation is not predictable, but is determined by conditions external to the digital computer.

Time-sharing Interrupts

The recently evolved time-sharing computers have the most stringent time requirements of all computer systems. In the inter-active time-sharing systems, where many users are on-line simultaneously, each user operates some type of console as though he had complete control of the computer. The timing of computer operations in this system must be quite efficient to give each user a satisfactory response time. Interrupts form an important part of this system since many tasks are executed asynchronously. Because of the precise timing requirements mentioned above, the interrupt system must be as efficient as possible.

Interactive users in a time-sharing computer system should have as much flexibility as possible in software and hardware features. The only hardware limitations that should be imposed upon them are basic system capability and the necessity for prevention of interference between users. If two or more jobs are being performed simultaneously, then any one of them must be prevented from stopping the computer, or placing it in a position so that it is uninterruptible. In addition to these mandatory requirements, other hardware features can be implemented to greatly enhance the efficiency of time-shared operation, and thereby reduce response time.

Each user is normally allotted a certain amount of time known as the

time slice, for his exclusive use of the CPU. Since his program will probably be transferred from core memory to secondary storage and back again between successive time slices, the time slice should be long enough to allow the ratio of computation time to transfer time to be reasonable. If it is too long however, the user response time will be excessive. Where interrupts are used, they should have minimum overhead to maximize CPU efficiency.

Request interrupts are used in time-sharing systems to signal the end of the time-slice, to increase efficiency of I/O operations, to permit the user to communicate with his programs, and to allow the user program to request some common service from the control program. If a user attempts to interfere with another user's program or execute an illegal instruction, this can be efficiently signaled by a fault interrupt. Time-sharing is one of the special situations where traps can be used to great advantage for this purpose. The exclusive selectivity scheme can be used to allow the system to accept interrupts from each user's console only during that user's time slice.

A time-sharing system is said to be multiusage if different classes of tasks are performed simultaneously. If one or more of these tasks is classified as real-time, then the system presents the greatest challenge to the systems designer. Every resource at his disposal must be analyzed and implemented with maximum efficiency, not the least important of which is the chosen interrupt system. [3]

SECTION IV

PROPOSED DIGITAL CONTROL LABORATORY TIME-SHARING SYSTEM

The new computer system for the Digital Control Laboratory (DCL) at the Naval Postgraduate School will consist of a standard Scientific Data Systems 930 computer with the optional equipment listed in Table 1.[4]

I. SDS 930 COMPUTER FEATURES

The SDS 930 is a fast, small, flexible, second generation computer, but one which was developed late in the second generation, and therefore contains features which are more advanced than those of earlier machines.

Time-multiplexed Communication Channel

The first of these features is the time-multiplexed communications channel (TMCC), which can, with interlace, handle I/O at speeds of up to 285,714 words per second depending upon the capability of the particular peripheral device with which the computer is exchanging information. The TMCC may communicate with up to 30 peripheral devices at the same time, but the transmission rate is reduced considerably by using more than one device at a time. The computer may contain up to four TMCC's, each with or without interlace. The interlace feature provides the CPU with the ability to transfer information by simply setting a pointer and counter, then initiating the transfer. The information exchange starts at the location specified by the pointer and transfers the number of words specified by the counter, all without further intervention by the CPU. Furthermore, the CPU is normally notified by the TMCC when the transfer has been completed by an I/O interrupt. It should be pointed out, however, that for each word transferred, the CPU is not

Quantity	Description
1	Time-Multiplexed Communication Channel
2	8,192 Words (24 bits) of Core Memory
1	Memory Interlace Control Unit
1	Real-Time Clock
1	Interrupt Control System
6	Priority Interrupt, two levels
2	Multiple Access to Memory
1	Twelve-Bit Option
1	Paper Tape Reader/Punch
1	Keyboard/Printer (Teletype)
1	Card Reader (200 card/minute)
1	Unbuffered Line Printer (140-160 lines/min.)
2	Magnetic Tape Transport (75 ips, 200 cpi)
1	Magnetic Tape Controller
1	Rapid Access Data Storage Unit
2	High Performance Display Console
1	Memory Interface Controller
2	CRT Channel
2	Analog Data Channel
1	Analog to Digital Converter
1	Digital to Analog Converter
1	Analog Computer

Table 1. Digital Control Laboratory Optional Equipment

allowed to process instructions for the duration of the transfer, which is 3.5 usec. The advantage is gained by the fact that single word transmission of the first data word requires a minimum of 19.25 usec, and each word thereafter, in sequence in memory to the same device, requires a minimum of 8.75 usec.

Programmed Operator

The next major feature is the programmed operator, or POP. If bit position two of an instruction contains a one, the operation code (bits three through eight) is not interpreted normally, but the program jumps to the address in memory specified by bits two through eight. This allows the programmer to design up to 64 macroinstructions and to gain direct access to them through branch instructions located in cells 0100 through 0177. Control will automatically be returned to the main program upon completion of each macroinstruction.

Multi-level Priority Interrupt System

One of the most important advanced SDS 930 features is the multi-level priority interrupt system. The computer is capable of handling up to 922 separate priority interrupt levels, two of which are standard equipment. The interrupts are divided into two basic classes: interval and external.

Internal interrupt. The internal interrupt is concerned with conditions within the computer and its standard peripherals which cause the program in progress to cease in deference to the appropriate interrupt subroutine. There are 26 possible internal interrupts. The two which are standard equipment are associated with the standard TMCC. There are two more optional interrupts for each of the other seven optional communications channels. The other four internal interrupts are used for the

power fail-safe option and the real-time clock. The power OFF interrupt causes the program in progress to stop, and the contents of all important registers to be stored in a specific location in memory whenever the voltage drops below a certain level. The power ON interrupt causes these register contents to be restored to the appropriate registers when the voltage again rises above this same level. The real-time clock can be used to cause interrupts at the expiration of time periods of specific length, or at specific times of the day.

External interrupt. The computer has a maximum capacity of 896 optional external interrupts. All external interrupts are lower in priority than any internal interrupt, but otherwise they operate in the same manner.

Normal interrupt. Each separate interrupt has associated with it a specific cell in core memory. When an interrupt occurs, the computer halts computation at the end of the present instruction and automatically executes the instruction in the memory cell associated with the appropriate interrupt. The instruction used in an interrupt location is determined by whether that level is a normal or single-instruction interrupt. A hardware change is required to convert a normal interrupt level to single-instruction operation. When a normal interrupt is used, the memory cell associated with it must contain a Mark Place and Branch instruction, BRM. This instruction, which is similar to the return jump instruction, causes transfer of control to the interrupt subroutine. The BRM "marks place" by storing the contents of the program counter (which contains the address of the next instruction in the main program) in the operand of the BRM, then executing the instruction contained in the following address. At the end of the subroutine, an Unconditional Branch instruction,

or BRU, is executed using indirect addressing. The operand address of this BRU is the first word in the interrupt subroutine, which in turn contains the address of the next instruction in sequence in the main program. In addition, it also automatically clears the interrupt.

Single-instruction interrupt. The single-instruction interrupt level allows execution of only the instruction in the interrupt location. Upon completion of this instruction, control is returned to the program previously being executed by calling the next instruction indicated by the program counter. The program counter is not altered by the housekeeping associated with the execution of the single interrupt instruction, nor by the execution of any instruction in this mode. As an example, the execution of a Reduce Memory, Skip if Negative instruction, SKR, in the CLOCK PULSE interrupt location, cell 075, causes the following special sequence. The operand is decremented by one, the interrupt is cleared, and control is returned to the main program. The SKR normally skips one instruction if the operand is negative after having been decremented, but in this special situation, if a negative count occurs, the skip is inhibited and a CLOCK SYNC interrupt is initiated. This interrupt causes execution of the instruction in cell 074. This is the method used to interrupt a program after a given elapsed time. An integer equal to the number of 16.67 msec intervals desired minus one is entered in the operand of the SKR instruction. When this cell becomes negative, a normal interrupt occurs which signals the end of the time period.

Selectivity. The interrupt system is equipped with the general allow/defer selectivity feature, which for this computer is called enable/disable. The general allow/defer is under program control, but the programmed defer may be overridden by a manual allow switch available to

the operator.

The individual recognize/ignore, here called arm/disarm, is available as an option, but is not included in the DCL system. However, the two real-time clock interrupts are equipped with a group recognize/ignore under program control. [5]

II. SDS 930 COMPUTER OPERATION

The following paragraphs will describe generally how the CPU functions, the logic of the interrupt system, and what effect the interrupt system has on CPU operation when an interrupt occurs.

Central Processing Unit Operation

Each machine instruction is executed during an integral number of 1.75 usec machine cycles. Each cycle contains eleven timing pulses, designated T8,T7,....,T0,Tr,Tp. These pulses are generated by a six flip-flop pulse counter (Q1-Q6) which is fed by a 6.28 mHz crystal oscillator. During execution of any given instruction, each machine cycle is assigned one of eight possible phases, Ø0,Ø1,....,Ø7. The phases are determined by a three stage counter (F1-F3) which is advanced to the next appropriate phase, usually by the Tp timing pulse, depending upon which instruction is being executed. Each action taken by the computer is triggered by the presence of a specific timing pulse during a specific phase of an instruction.

The 930 word is 24 bits long. The instruction format is as indicated in Figure 1. The CPU contains the following 24 bit registers: an exchange register, C, an accumulator, A, an accumulator extension register, B, and an index register, X. The program address counter, P, and storage address register, S, are 15 bits in length; the operation code register, O, contains 6 bits. These registers generally operate like

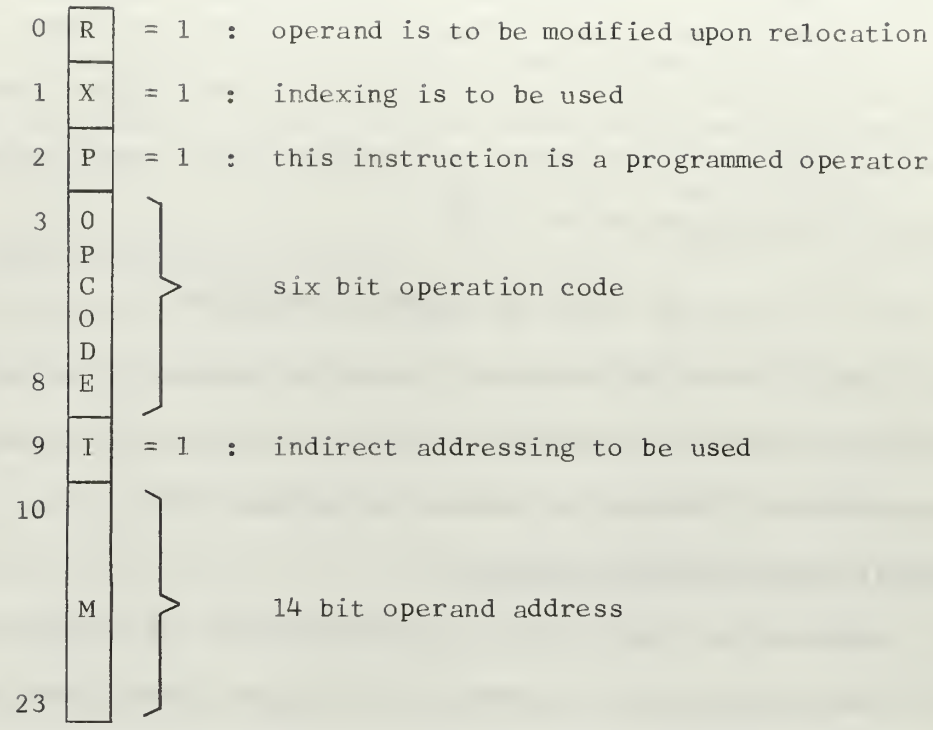


Figure 1. Instruction Format

those of any normal second generation computer, except that data is transmitted between the 24 bit registers in a serial-octal mode, or three bits at a time. The sending and receiving registers both shift three bit positions during each transmission, and the process continues until each register has been shifted 24 bits. The use of clocked flip-flops allows information to be exchanged between two registers without intermediate storage since old information may be transferred out of a flip-flop at the same time that new information is is inserted without mutual interference.

Certain operand addresses are transmitted from one register to another, in serial-octal mode, through a three stage full adder. As an example, when a branch instruction is executed, the operand address in C is sent to P, three bits at a time, through the full adder. The three

least significant bits of C (C21-C23) form the augend inputs to the adder, the addend inputs are disabled, and the output sum (Add1 - Add3) goes to the three most significant bits of P (P0-P2). If indexing is used in a branch instruction, the addend input from X is enabled, thereby allowing the contents of the index register to be added to the address as it is being transferred. This allows indexing to be performed with no increase in instruction time.

The path for all words to and from memory uses parallel transmission between C and the memory data register, M. Other data transfers which use parallel transmission are: the opcode from C to O, and operand addresses from C or P to S for storage reference.

The last phase in any instruction is designed as the End phase in addition to be assigned a number. When an instruction other than a branch or skip is executed, housekeeping to prepare for the next instruction, NI, is performed during End. This housekeeping normally consists of the following operations: (1) P is incremented by one to reference the next instruction in sequence, (2) P is transmitted to S to provide storage reference, (3) the contents of the address in S is loaded from memory into M, (4) the instruction in M is transmitted to C, and (5) the phase counter is reset to Ø0. The next phase will then be the first phase, or Ø0, of NI. The first timing pulse of this phase, or T8 Ø0, transfers the opcode from C to O. The instruction being executed then determines the action to be taken by the CPU. [6]

Interrupt Control System Operation

Normal interrupt. The logic for level 34 of the interrupt control system, ICS, is shown in Figure 2. An interrupt request for this level must consist of a one (+4 volts) at (I34). This request should be held

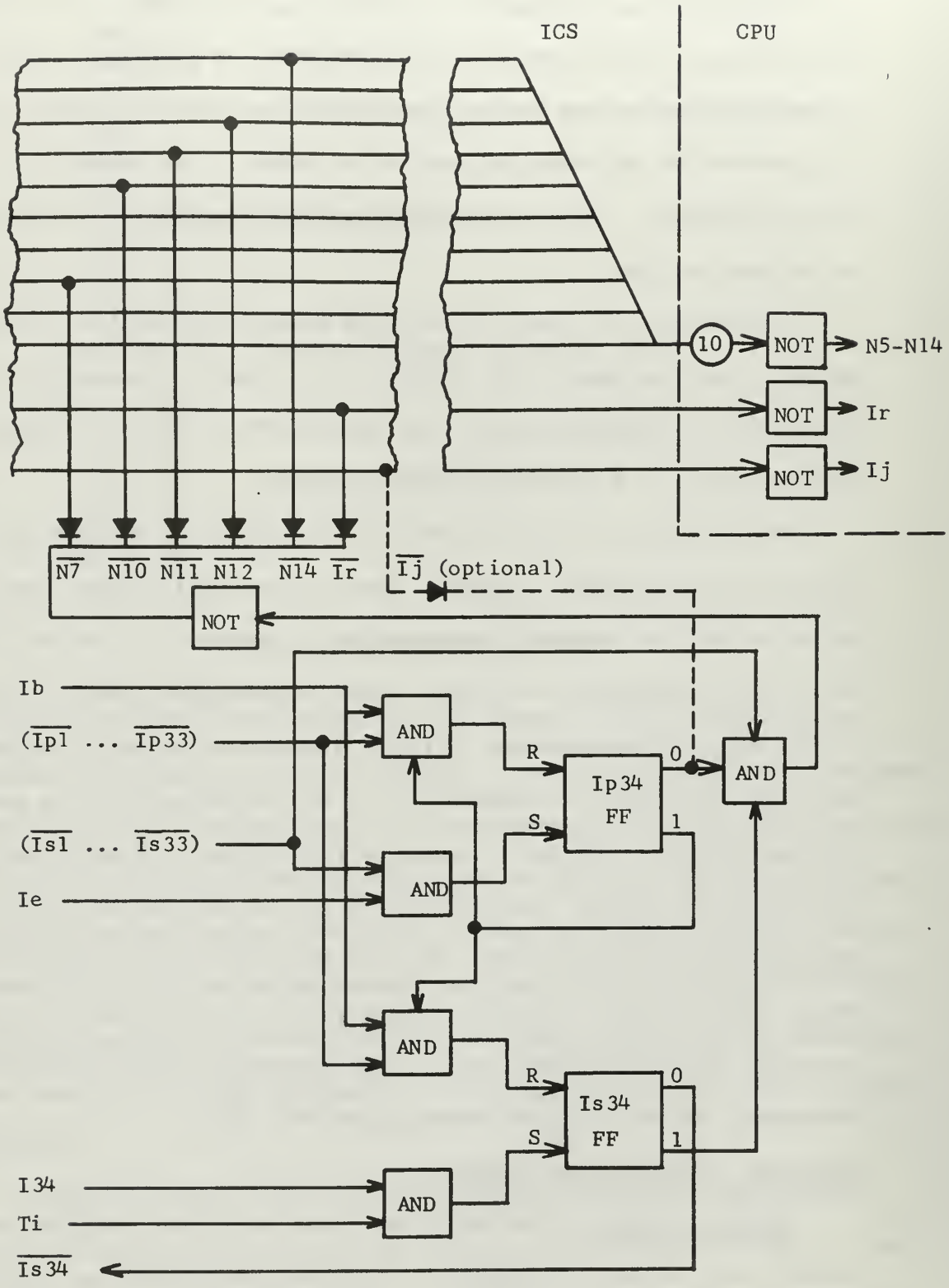


Figure 2. Interrupt Control System Logic, Level 34

in the one state for at least 1.75 usec, then cleared as soon as possible. When (I34) is true, Is34, the level 34 interrupt storage flip-flop, will be set by the trailing edge of the next TO timing pulse. Is34 indicates, when set, that an interrupt has been received on level 34 that has not been processed. If no higher level interrupt is waiting or being processed, the interrupt signal, Ir, and the storage address unique to level 34, 0235, are sent to the CPU. If the interrupt system has been enabled, the present instruction is not an EOM, and an interlaced I/O operation is not taking place; then Ir sets Int, the interrupt flip-flop in the CPU. The necessary conditions to set Int (designated sInt) are shown in the following logic equation:

$$sInt = \overline{(\emptyset 5\ 01\ 05\ \overline{Ts}\ Q1)}\ T4\ End\ \left[Is + Ir\ (En + \textcircled{En})\right]\ \overline{Ts}. \qquad (1)$$

Once Int is set, the normal signal which causes P to be transmitted to S, Sxp, at End T3 is inhibited:

$$Sxp = T3\ \overline{Int}\ (End + Ju\ \overline{Eax})\ Go + T3\ \textcircled{Kmc}. \qquad (2)$$

Instead, a signal, Sxn, is generated to transmit the interrupt address, N5-N14, to S:

$$Sxn = T3\ Int\ \overline{Inr}. \qquad (3)$$

In the example of an interrupt to level 34, Sxn causes the address 0235 to be transmitted from the N cable receivers to S. Inr, the interrupt interlock flip-flop, is always set six timing pulses after Int is set:

$$sInr = Tp\ Int. \qquad (4)$$

Inr prevents Sxn and other interrupt initiate signals from being generated during subsequent phases.

The next timing pulse after sInt causes an interrupt recognition signal, Ie, to be sent to the ICS:

$$Ie = Int\ \overline{\overline{Inr}}\ Ti, \tag{5}$$

where Ti is true from T3 through T0:

$$Ti = Q6\ Q1. \tag{6}$$

Ie sets Ip34, the level 34 interrupt process flip-flop, if no higher level interrupt is in the waiting or active states. Ip34 indicates that the CPU is now processing the interrupt subroutine initiated by this level, and disables Ir and N5-N14.

The CPU now proceeds to execute the BRM which must be located in 0235. This instruction transfers control to the appropriate subroutine as previously described. During Ø0 of the BRM, Int and Inr are reset (designated rInt) so that normal CPU operation may take place while executing the interrupt subroutines:

$$rInt = rInr = T7\ \overline{Sk}\ Go + (St). \tag{7}$$

Otherwise Int would inhibit the transfer of the address of NI from P to S.

The indirectly addressed BRU at the end of the subroutine, in addition to returning control to the main program, also sends an interrupt termination signal, Ib, to the ICS:

$$Ib = Ø0\ Ia\ \overline{01}\ (\overline{02}\ \overline{04}\ \overline{05}\ 06)\ \overline{Ts}\ Ti + Ij\ \overline{Ts}\ End\ \overline{Inr}\ Ti. \tag{8}$$

Ib resets the process and storage flip-flops, provided that no higher level interrupt is in progress. This indicates that the interrupt has been returned to the inactive state. If (I34) is still true during the

phase after Ib is received by the ICS, the interrupt will be repeated. The optimum pulse length for an interrupt signal is therefore 1.75 usec.

Single-instruction interrupt. A normal interrupt may be converted to a single-instruction interrupt by inserting a connection as indicated by the dotted line in Figure 2 to furnish the single-instruction interrupt signal, Ij. Any instruction requiring two or more machine cycles may be used in the memory cell assigned to a single-instruction interrupt, but the normal use of this type of interrupt uses an SKR instruction to decrement a counter.

When a normal skip instruction is executed, P is increased by one by the P + 1 adder during the next to last phase in order to address the next instruction. In any phase where a P + 1 is possible, P is right circular shifted five times, three bits at a time. P is increased by one if the indirect address flip-flop, Ia, is set when this shift takes place. During this same phase the condition which could cause the skip is checked. If satisfied, Ia is set during the last phase by the skip flip-flop, Sk, to cause P to be increased by one again, thereby causing NI to come from cell P + 2 rather than P + 1. For an SKR instruction, Sk is set during phase four if M is negative:

$$sSK = \ldots + \emptyset 4\ 01\ \overline{05}\ \overline{06}\ Tr\ C\emptyset + \ldots\ , \tag{9}$$

where CO is the sign bit of C which is enabled when C contains a negative number. Sk in turn sets Ia during End (Ø7):

$$sIa = \ldots + T8\ \emptyset 7\ Sk\ \textcircled{Kr}\ (\overline{Ij} + Inr)$$
$$+T8\ (\emptyset 4 + \emptyset 6)\ \textcircled{Kr}\ (\overline{Ij} + Inr) + \ldots \tag{10}$$

and the skip is effected.

When SKR is executed as a single-instruction interrupt, its operation is altered. During Ø4 and Ø7, the presence of Ij causes (Ij + Inr) in Eq. 10 to be false, since Inr is true only during Ø0, and P is not modified during the SKR execution. Control is therefore returned to the next instruction in sequence in the main program upon completion of the interrupt. The single-instruction interrupt termination is caused by the presence of Ij in the second term of Eq. 8, and occurs regardless of the instruction executed.

Real-time clock interrupts. The real-time clock uses two special interrupts: one normal and one single-instruction. The A-C line voltage is stepped down to 10 volts peak-to-peak and clipped to provide a square wave as a timing signal, Rtc, to the real-time clock control circuit. This control circuit consists of three flip-flops, Rt1, Rt2, and Rt3. Rt1 is set under program control when the real-time clock is armed. Rt2 is set at the first Ti during the positive half of Rtc:

$$sRt2 = \overline{Rt2}\ Rtc\ Ti. \tag{11}$$

When Rtc goes to zero one half cycle later, Rt3 is set:

$$sRt3 = Rt1\ Rt2\ \overline{Rt3}\ \overline{Rtc}\ Ti, \tag{12}$$

but remains set for only one phase:

$$rRt3 = Rt3\ Ti + St. \tag{13}$$

Rt3 thus serves as a 1.75 usec interrupt request, CLOCK PULSE, which occurs every 16.67 msec (1/60 sec); it sets Is8 at Ti time to trigger the single-instruction interrupt located in cell 075. If this instruction is an SKR, it operates as previously described, except that the

following signal is generated if the effective memory location is reduced to a negative value:

$$Ski = Sk\ \emptyset 7\ Ij\ \overline{Inr}. \tag{14}$$

Ski is the CLOCK SYNC interrupt request to Is7 which signals that the elapsed time interval set in the effective memory location is completed. This normal interrupt, located in cell 074, causes the desired action to be taken at the end of the interval.

Both real-time clock interrupts are always enabled, since the interrupt signal they send to the CPU is Is rather than Ir. As shown by the bracketed term of Eq. 1, only Ir requires that the ICS be enabled by the program controlled enable flip-flop, En, or the console enable switch, (En). [7]

III. DIGITAL CONTROL LABORATORY SYSTEM OBJECTIVES

The basic objective of the DCL installation is to simultaneously perform in the priority order indicated (1) hybrid computation in conjunction with a Comcor analog computer, (2) interactive program composition, checkout and execution at two display consoles, and (3) background batch processing of jobs with one of two possible priorities. In consideration of these tasks and the equipment available in the system, several hardware modifications involving interrupts are proposed for future implementation into the DCL system. These modifications are (1) a mode of operation for interactive users and batch processing in which restrictions are placed on the instruction repetoire and access to certain protected portions of memory, (2) the mechanism for detecting illegal instructions and (3) memory protect violations while in this mode, (4) a second real-time clock which signals completion of a user's time quantum,

and (5) the circuitry to allow sequences of normally non-interruptible instructions to be interrupted.

Purpose of Modifications

The following paragraphs will analyze the proposed overall system philosophy and the reasons for each modification.

The hybrid computation has highest priority because it is a real-time process. In a real-time system, since the computer must be able to respond to external stimuli fast enough so that information is not lost and/or that answers are provided to the external equipment in time to be of value, all display console and batch computation should be performed during hybrid computation idle time. When the CPU is not involved in hybrid computation, the execution control system should transfer control to one of the two console users on a round robin basis. If any idle CPU time remains, it should be scheduled for use by the background batch process.

Multiple operating modes. The assumption is made that all programming for the executive control system and hybrid program will be sufficiently error-free so that computer hangups and mutual interference will not occur. It is also assumed that these two programs will reside in a protected area in memory at all times; the control system must remain there to retain control, and the hybrid program must remain there to provide adequate response time. These two programs will therefore operate in the executive, or normal, mode as distinguished from the user mode.

When the computer is in the executive mode, its operation will be identical in all respects with the standard SDS 930. This is necessary so that all standard software will be useable without modification. The executive control system will shift to user mode under program control

at the start of the user time quantum. Control will shift back to executive mode automatically when (1) the time quantum is completed, (2) an interrupt occurs, (3) a privileged instruction is executed, or (4) a protected memory cell is referenced.

Privileged instructions. Since timing is so critical in a time-sharing system, each user must be restricted in his use of instructions. This implies, in effect, that a user may use his time to perform useful computation, but he must be prevented from "wasting" time. Therefore, all I/O operations must be performed by the executive control program, which can be programmed to do I/O very efficiently. The user must be prevented from inadvertently executing I/O instructions, as well as halt and illegal instructions. The execution of any of these would waste time. The privileged instruction modification is proposed to prevent the user from executing these instructions.

Memory protection. The user must also be prevented from destroying the executive control program or the hybrid program. The only positive means of accomplishing this is to add a feature that will stop the user before he executes an instruction which references illegal memory and to notify the executive control program when this happens. The executive can in turn notify the user of his error so that it may be corrected. In the meantime, continuity of computer operation is maintained.

Quantum timer. Since the system real-time clock will be an integral part of the hybrid system and will probably be used to provide absolute time for most problems, a second clock, programmed as an interval timer, is necessary to interrupt a console user at the end of his time quantum so that control may be shifted to the other user console.

Non-interruptible sequences. The computer system can be effectively disabled by the execution of a long or infinite chain of indirect addressing

or Execute instructions. Neither chain is interruptible, so a user may gain complete control of the system in this manner. To prevent this, the non-interruptible sequence modification permits such chains to be aborted and interrupted when the computer is in user mode.

Some of these modifications are similar to features of the SDS 940 computer system. This system, designed at the University of California, Berkeley, is a time-sharing version of the SDS 930 which can accommodate 32 interactive users. The features which are important from the standpoint of time-shared operation are a memory map used in conjunction with a 64K word magnetic core memory, a large rapid access data storage unit for secondary storage of user programs, a specially designed asynchronous communications controller, three modes of operation, system programmed operators, privileged instructions and an instruction interruptibility feature. [8]

The proposed modifications can be implemented using SDS T Series logic modules. These modules are available in flexible configurations of both positive and negative logic and are fully compatible with SDS 930 circuitry. [9]

IV. PROPOSED LOGIC DESIGN CHANGES

The remainder of this section describes the proposed modifications as changes to the SDS 930 logic equations. New logic terms in existing equations, and new equations are indicated by underlining.

Multiple Operating Modes

Method. The mode of operation is determined by the state of the mode flip-flop, Mo:

$\overline{Mo}$ = Executive mode

Mo = User mode.

All of the changes and additions to the SDS 930 logic will be enabled by Mo.

The transition from executive to user mode will be effected by the execution of the following EOM instruction:

0 02 20040.

When this instruction is executed, Mo will be set:

$$sMo = Eom\ C10\ \overline{C11}\ C18\ T0, \tag{15}$$

where

$$Eom = (Ø5\ \overline{01}\ 05\ Ts\ \overline{(\overline{Q2}\ \overline{Q5})}\ 04) + \ldots, \tag{16}$$

which is the enable for an EOM instruction. A branch instruction to a location in the users program area must immediately follow the EOM; otherwise a memory protect violation will occur. Retrieval of the branch instruction itself from memory will not cause a violation for reasons to be explained later.

Logic. The logic required for this modification is one flip-flop and four NAND gates.

Privileged Instructions

Method. When the computer is operating in the user mode, instructions which halt the computer or reference external equipment, and instructions with illegal SDS 930 opcodes are called privileged instructions. Execution of a privileged instruction causes a trap to occur. A trap as defined here can interrupt another interrupt of any priority, and its execution does not in any way inhibit other interrupts or traps from being recognized and executed immediately when received. Privileged instructions are listed and identified in Table 2.

When a privileged instruction trap occurs, the privileged instruction is converted to a NOP, and the instruction in memory cell 040 is

Opcode	Definition	Opcode	Definition
00	Halt	24	Illegal
02	Energize Output Multiplex	25	Illegal
03	Illegal	26	Illegal
04	Illegal	27	Illegal
05	Illegal	30	Channel Y into Memory
06	Energize Output Direct	31	Illegal
07	Illegal	32	Channel W into Memory
10	Memory into Y Channel	33	Parallel Input
11	Illegal	34	Illegal
12	Memory into W Channel	40	Skip is Signal Not Set
13	Parallel Output	42	Illegal
15	Illegal	44	Illegal
21	Illegal	45	Illegal
22	Illegal	47	Illegal

Table 2. Privileged Instructions

executed. This trap instruction should normally be a BRM, which will store the address of the privileged instruction and branch to a subroutine which will inform the user of the violation.

Execution of a privileged instruction generates a privileged instruction decode, Pid:

$$\begin{aligned} Pid = {} & \overline{C3}\ \overline{C5}\ C6 \\ & + \overline{C3}\ C5\ \overline{C6} \\ & + \overline{C4}\ \overline{C5}\ \overline{C6}\ \overline{C8} \\ & + \overline{C3}\ \overline{C4}\ \overline{C5}\ C7 \\ & + \overline{C4}\ \overline{C5}\ \overline{C7}\ \overline{C8} \\ & + \overline{C4}\ \overline{C5}\ C6\ C8 \\ & + \overline{C3}\ \overline{C4}\ C5\ \overline{C7}\ C8 \\ & + \overline{C3}\ C4\ \overline{C5}\ \overline{C7}\ C8 \\ & + \overline{C3}\ C4\ C5\ \overline{C7}\ \overline{C8} \\ & + \overline{C3}\ C4\ \overline{C5}\ C7\ \overline{C8}. \end{aligned} \qquad (17)$$

Pid is enabled from C rather than O so that it will be present at T8 Ø0, which is the timing pulse that causes transmission of the opcode from C to O. If the computer is operating in user mode and indirect addressing is not being used, Piq is generated:

$$Piq = Pid\ Mo\ Go\ C2\ Ø0\ T8\ \overline{Ia}. \qquad (18)$$

Piq sets the privileged instruction flip-flop, Pi:

$$sPi = Piq, \qquad (19)$$

and prevents initiation of the normal instruction execution. The transfer of the opcode from C to O is inhibited by Piq:

$$Oxc = Ø0\ T8\ \overline{Ia}\ Go\ C2\ \overline{Piq\ Ob\ Ai}\ . \qquad (20)$$

Oxc is always preceeded by a clear O signal, Oc, during the last timing pulse of the End cycle:

$$Oc = Tp\ End\ \overline{Sk} + \cdots . \tag{21}$$

Oc clears all stages of O except 02, which it sets:

$$s02 = Oc. \tag{22}$$

Clear O therefore actually sets O to 020, which is the opcode for a NOP instruction. Under normal conditions if an opcode whose second bit position contains a zero is sent to O, a zeros transmission occurs:

$$r02 = Oxc\ \overline{C4}. \tag{23}$$

The first phase of the privileged instruction, Ø0, has now been converted to the first (and only) phase of a NOP. Normally, Ø0 of NOP is immediately converted to Ø5:

$$sF1 = Ø0\ T8\ \overline{Ia}\ \overline{C2}\ \overline{C5}\ \overline{C8}\ (\overline{C3} + \overline{C4}) + \ldots \tag{24}$$

$$sF3 = Ø0\ T8\ \overline{Ia}\ \overline{C2}\ \overline{C5}\ \overline{C8}\ (\overline{C3} + \overline{C4}) + \ldots \tag{25}$$

Since this special NOP opcode was never in C, the above transition does not occur, and the phase counter must be forced to Ø5:

$$sF1 = \ldots \underline{+ Piq + T8\ (Ob + Ai)} + \ldots \tag{26}$$

$$sF3 = \ldots \underline{+ Piq + T8\ (Ob + Ai)} + \ldots . \tag{27}$$

During execution of an instruction which does not reference memory, P is normally incremented by one (P + 1) during the first phase to obtain NI. In this case NI will be a trap instruction, so Piq inhibits sIa which in turn prevents P + 1:

$$sIa = \ldots + \left[Ø0\ \overline{Ia}\ T8\ \overline{C2}\ \overline{C5}\ \overline{C8}\ (\overline{C3} + \overline{C4})\right] \textcircled{Kr}\ \underline{\overline{Pi}\ \overline{Ob}\ \overline{Ai}\ \overline{Piq}} + \ldots \tag{28}$$

P + 1 is inhibited so that the address of the privileged instruction

will be in P when the trap occurs.

The only other significant task of NOP is to transfer the address of NI from P to S and initiate a memory read operation. This transfer must be inhibited by Pi:

$$Sxp = T3\ \overline{Int}\ (End\ \underline{\overline{Pi}\ \overline{Ob}} + Ju\ \overline{Eax})\ Go. \quad (29)$$

Since S is cleared at T4:

$$Sc = T4\ (End + \overline{F1}\ \overline{F2})\ \overline{Inr} + \ldots\ , \quad (30)$$

the underlined portion of the following equation sets S equal to 040:

$$\begin{aligned} sS9 = {} & Sxc\ C6 \\ & + Sxp\ P6 \\ & + Sxn\ N9 \\ & + Sx48 \\ & \underline{+ T3\ (Pi + Ob)}\ . \end{aligned} \quad (31)$$

Since cell 040 is in protected storage, the computer must be shifted from user to executive mode at T3 to prevent a user mode memory violation:

$$\underline{rMo = T3\ (Pi + Ob) + \ldots}\ . \quad (32)$$

Pi is cleared at T2 to remove all indications that a trap has occurred:

$$\underline{rPi = Pi\ T2}\ . \quad (33)$$

At Tp, the BRM in cell 040 is transferred from the memory register to C, and is executed during the next phase. The BRM stores the contents of P in the operand, M, and takes the contents of M + 1 as NI.

Logic. One flip-flop and 17 NAND gates are required to accomplish this modification.

Memory Protection

Method. Certain portions of memory may be designated as protected areas by the computer operator. These protected areas may be manually

selected by an eight-position, three-level rotary switch. Each switch position decodes to a different combination of three signals, $\overline{Mp1}$ - $\overline{Mp3}$, as shown in Table 3. This flexibility in the size of protected memory is provided to allow for changes in size of the executive control system and the hybrid program.

If an instruction attempts to read out of, write into, or execute a subsequent instruction from protected memory, the computer will revert to executive mode and trap to location 041.

A memory reference is normally made twice during instruction execution. The first occurs during ØØ when the instruction carries out the appropriate operation with the operand; the second reference, which always occurs during an End phase, is made to retrieve NI. These two references are the same for a normal branch instruction, since the operand is taken as NI. Some instructions use the operand address for purposes other than memory access, and therefore reference memory only once.

Switch Position	Decoded Output			Protected Memory Addresses	User Memory Available
1	$\overline{Mp1}$	$\overline{Mp2}$	$\overline{Mp3}$	None	16K
2	$\overline{Mp1}$	$\overline{Mp2}$	$Mp3$	00-03777	14K
3	$\overline{Mp1}$	$Mp2$	$\overline{Mp3}$	00-07777	12K
4	$\overline{Mp1}$	$Mp2$	$Mp3$	00-013777	10K
5	$Mp1$	$\overline{Mp2}$	$\overline{Mp3}$	00-017777	8K
6	$Mp1$	$\overline{Mp2}$	$Mp3$	00-023777	6K
7	$Mp1$	$Mp2$	$\overline{Mp3}$	00-027777	4K
8	$Mp1$	$Mp2$	$Mp3$	00-033777	2K

Table 3. Memory Protect Switch Positions

If S contains an address in protected memory at any time, a protect memory enable, Pme, is generated:

$$\begin{aligned} Pme = {} & (Mp1)\ S1 \\ & + (Mp1)(Mp2)\ \overline{S2} \\ & + (Mp2)\ \overline{S1}\ \overline{S2} \\ & + (Mp1)(\overline{Mp2})(Mp3)\ \overline{S2}\ \overline{S3} \\ & + (Mp1)(Mp2)(Mp3)\ S2\ \overline{S3} \\ & + (\overline{Mp2})(Mp3)\ \overline{S1}\ \ \overline{S2}\ \overline{S3} \\ & + (Mp2)(Mp3)\ \overline{S1}\ \ S2\ \overline{S3}\ . \end{aligned} \tag{34}$$

Since S is always cleared at T4 and set to the appropriate memory address at T3, an illegal address detected at T2 will be used to set the out-of-bounds flip-flop, Ob:

$$sOb = Pme\ Mo\ End\ T2\ \overline{Int} + Pme\ Mo\ \emptyset 0\ T2\ \overline{RCH}\ \overline{NOP}\ \overline{EAX}\ \overline{(BRX\ \overline{Xw1})}\ , \tag{35}$$

where the capitalized terms are enables for the indicated instructions:

$$RCH = 01\ \overline{02}\ \overline{03}\ 04 \tag{36}$$

$$NOP = \overline{01}\ 02\ \overline{03} \tag{37}$$

$$EAX = 01\ 02\ 03\ 04\ 05\ 06 \tag{38}$$

$$BRX = Ju\ 01\ \overline{05} \tag{39}$$

The Register Change instruction, RCH, and NOP are the only two non-privileged instructions which do not use the operand address for a memory reference. The Copy Effective Address into Index Register instruction, EAX, reads the contents of M, but never uses it. The Increment Index and Branch instruction, BRX, only branches if bit nine of X contains a one. If this bit, which is in Xw1 at T2, is a zero, the branch is inhibited. In this case $\overline{BRX\ \overline{Xw1}}$ would prevent Ob from being set,

since no reference is made to the protected location.

If Ob is set at End T2, transfer of NI from C to O is inhibited by Ob, as indicated in Eq. 20, and O is set to 020 by Oc:

$$Oc = \cdots + Tp\ (Ob + Ai)\ . \qquad (40)$$

If Ob is set at Ø0 T2, the present instruction is aborted by Oc (Eq. 40) and a NOP occurs during the following phase. If indexing is used in the aborted instruction, the illegal address determination is made using the effective address after indexing, since the address portion of X is added to the address portion of C during Ø0 T7-T3.

If the aborted instruction is one which normally writes information into memory, the operand is protected by Ob, which inhibits the transfer from C to M:

$$Mxc = \overline{Ob}\left[\overline{Tsm}\ (Ø4 + Eax + Ju\ 01\ 05) + Tsm\ \overline{R9} + \overline{Kmc}\right]\ . \qquad (41)$$

Mxc is also an enable for a memory write operation. Its absence converts the memory reference from a clear-write cycle to a read-restore cycle. The operand is therefore read from memory into M and restored to memory without modification.

The operation of the memory protection NOP is similar to that of the privileged instruction NOP in that Ob (1) advances the phase counter to phase five (Eq. 26 and 27), (2) inhibits the address transfer from P to S (Eq. 29), (3) reverts to executive mode (Eq. 32), and (4) sets S to the trap address, 041, by Eq. 31 and:

$$sS14 = \ldots + T3\ Ob. \qquad (42)$$

Changes to the memory protection trap operation must be made for the BRM instruction. Since BRM stores the contents of P in the operand, it exchanges the contents of P and C10-C23 during the first phase. If Ob

is set at Ø0 T2, this indicates that the initial contents of P are about to be stored in an illegal address. If the BRM is aborted as described above, the P will contain the operand address of the offending instruction rather than its location. This situation is corrected by the following logic changes.

At the same time that Oc sets 02, a signal is generated to set 06:

$$s06 = Oxc\ C8\ \underline{+\ Oc\ Ob\ Ju\ 01\ 05}\ , \tag{43}$$

which causes the address portion of C to be transferred back to P. C and P are not exchanged at this point, since the operand address is not only no longer needed, but it may have been changed if indexing was used. Since P is shifted during the NOP by Pr3, the transfer is effected by shifting C:

$$Cr3 = \ldots\ \underline{+\ Ø5\ Ob\ 06\ \overline{Ts}\ Q1}\ +\ \ldots\ , \tag{44}$$

and enabling the transfer from the full adder to P:

$$sPO = Pr3\left[Add1\ (\ldots\ \underline{+\ Ob\ 06})\ +\ \ldots\right]\ , \tag{45}$$

$$rPO = Pr3\left[\overline{Add1\ (\ldots\ \underline{+\ Ob\ 06})\ +\ \ldots}\right]\ . \tag{46}$$

Similar terms are used in the equations for P1 and P2.

When executive control system executes the EOM to cause the transition to user mode, Mo is set at TO of the EOM. The retrieval of NI (the branch to the user area) does not cause a memory protect violation since sMo comes two timing pulses too late to cause Ob to be set (Eq. 35). However, the branch instruction must have a non-protected (user area) address to prevent a violation from occurring during its execution.

Logic. One flip-flop and 17 NAND gates are required to accomplish this modification.

Quantum Timer

Method. The user time quantum is measured by the second real-time

clock, which will be programmed as a subjective clock, or interval timer. This clock, known as the quantum timer, is triggered by an SDS T Series Medium Frequency Clock Oscillator, Model CT16. The output from the CT16, Qtc, is a four volt peak-to-peak, 8 kHz square wave. The quantum timer control circuit consists of four flip-flops, Qt1-Qt4, which convert Qtc to interrupt signals. Its operation is similar to that of the real-time clock control circuit, except that Qt1 and Qt2 are wired as toggle flip-flops in series to step down the frequency. The output of Qt2 is a 2 kHz square wave. Qt3 and Qt4 operate as follows:

$$sQt3 = \overline{Qt3}\ \overline{Qt2}\ Ti \tag{47}$$

$$rQt3 = Qt3\ Qt4\ Ti + (St) \tag{48}$$

$$sQt4 = Qt3\ \overline{Qt4}\ Qt2\ Ti \tag{49}$$

$$rQt4 = Qt4\ Ti + (St) \tag{50}$$

Qt4, a 1.75 usec pulse which occurs every 500 usec, is the request to the lowest priority external interrupt when in user mode:

$$(I24) = Mo\ Qt4\ . \tag{51}$$

The use of Mo here effectively ignores the quantum timer in executive mode.

Interrupt level 24 is wired as a single-instruction interrupt. The memory cell associated with this level, 0213, will contain an SKR instruction whose operand is a measure of the user time quantum. This operand is decremented every 500 usec while in user mode. When it becomes negative, it enables Sku:

$$Sku = Sk\ Ø7\ Ij\ \overline{Inr}\ Ip24\ , \tag{52}$$

which is the interrupt request to level 23. Since Ski would also be enabled at this time (Eq. 14), it must be modified to prevent the

expiration of the time quantum from triggering the CLOCK SYNC interrupt:

$$Ski = Sk\ Ø7\ Ij\ \overline{Inr}\ \underline{Ip8}. \tag{53}$$

If an interrupt occurs when the computer is in user mode, an automatic transition to monitor mode is made:

$$\underline{rMo = \ldots + Tr\ Mo\ Int}. \tag{54}$$

However, if the interrupt is of the single-instruction type, the computer must automatically return to user mode since a memory protect violation would otherwise occur when NI is referenced after the interrupt instruction. Retrieval of the interrupt instruction itself will not cause a memory protect violation since Int inhibits the setting of Ob at End T2 (Eq. 35). This automatic return is accomplished by the following procedure. The occurrence of a single-instruction interrupt in user mode sets Usi:

$$\underline{sUsi = Mo\ Int\ Ij\ Tr} \tag{55}$$

at the same time that Mo is reset. If Usi is set, the interrupt termination signal causes a shift back to user mode:

$$\underline{sMo = \ldots + Usi\ Ib}\ , \text{ and} \tag{56}$$

resets Usi:

$$\underline{rUsi = Usi\ Ib}\ . \tag{57}$$

Interrupt level 23 is the normal interrupt which signals completion of the time quantum. This interrupt operates like other normal interrupts. It can only occur in user mode, since it is triggered by a signal which is initially recognized by Mo. The interrupt subroutine for level 23, which will normally be used to swap users in and out of core, must contain and EOM instruction to shift back to user mode, if such a shift is desired.

Logic. One oscillator module, five flip-flops, and 11 NAND gates are required to accomplish this modification.

Non-interruptible Sequences

Method. Sequences of multilevel indirect addressing or Execute instructions, EXU, are normally not interruptible. The design changes described in the following paragraphs allow these sequences to be interrupted while in user mode. If an interrupt is received, the offending instruction is aborted, and an interruptible NOP is executed in its place. Upon completion of the interrupt subroutine, the aborted instruction is executed again from the beginning.

If an EXU or an indirect addressing instruction is being executed when an interrupt signal is received, the allow interrupt flip-flop, Ai, is set:

$$sAi = [Ir(En + \textcircled{En}) + Is]\ \emptyset 0\ T0\ \overline{Int}\ Mo$$
$$(Ia\ \overline{RCH}\ \overline{NOP} + \overline{01}\ 02\ \overline{03}\ \overline{04}\ 05\ 06)\ . \qquad (58)$$

Ai resets the indirect address flip-flop on the following timing pulse:

$$rIa = \ldots + Ai\ Tr\ . \qquad (59)$$

The NOP is forced at Tp (Eq. 40) and unwanted microoperations which may have been initiated by the setting of Ju and Eax from C are inhibited:

$$sJu = (\emptyset 0\ T8\ \overline{Ia}\ Go\ C2 + \emptyset 0\ T8\ \overline{Ia}\ Go\ \overline{C4}\ \overline{C5}\ C8\ \overline{C9}$$
$$+ \emptyset 0\ T8\ \overline{C9}\ \overline{02}\ \overline{03})\ \overline{Ob}\ \overline{Pi}\ \overline{Ai}\ \overline{Piq}\ . \qquad (60)$$

$$sEax = (\emptyset 0\ T8\ \overline{Ia}\ Go)\ C2 + (\emptyset 0\ T8\ \overline{Ia}\ Go)\ C3\ C4\ C5\ C6\ C7\ C8\ \overline{C9}$$
$$+ (\emptyset 0\ T8\ \overline{C9})\ 01\ 02\ 03\ 04\ 05\ 06\ \overline{Ob}\ \overline{Pi}\ \overline{Ai}\ \overline{Piq}. \qquad (61)$$

Oxc and sIa are inhibited by the same logic used for a trap forced NOP (Eq. 20 and 28); inhibiting the latter causes the aborted instruction to

be repeated after the interrupt is cleared because it prevents the P + 1 increment. The phase counter is also advanced to Ø5 (Eq. 26 and 27).

The only function performed by the NOP is to get interrupted. Retrieval of the interrupt instruction does not cause a memory protect violation because Int inhibits sOb (Eq. 35). Ai is reset at T3 and the computer is shifted back to executive mode at Tr of the NOP (Eq. 54).

If the interrupt was not single-instruction, the interrupt subroutine must contain an instruction to shift back to user mode. The determination of which mode was interrupted can be made by examining the operand address of the first instruction in the subroutine. If that address is not in protected storage, then the computer must have been in user mode.

Logic. One flip-flop and seven NAND gates are required to accomplish this modification. [10, 11]

V. SOFTWARE EQUIVALENTS

The following comparative analysis between the foregoing proposed modifications and the accomplishment of the same objectives with software is presented to demonstrate the desirability of these hardware modifications.

Multiple Operating Modes

The absence of hardware changes precludes the use of the two-mode concept. The user mode is implemented solely to allow the other modifications to apply only to the display console users and the batch processing jobs. There is no software equivalent to the user mode hardware.

Privileged Instructions

Detection and prevention of the execution of privileged instructions can be feasibly implemented only by allowing the user to program through a compiler. If the user is restricted to a conversational subset of FORTRAN, for example, the privileged instruction protection will automatically exist, since all such instructions will be excluded from the allowable object code. In addition, since each legal FORTRAN statement generates a legal, executable set of machine instructions which is free of errors, the object code will not contain inadvertent jumps into data areas with the usual disastrous results.

If the console user is allowed to use an assembly language, the assembler can restrict his use of machine instructions to any desirable subset. It cannot, however, prevent his executing a jump to an address which does not contain an assembled instruction. If this should happen, not only will the user's program be destroyed, but a privileged instruction execution is almost certain to result.

Memory Protection

Memory protection can be provided for user programs only by restricting his operations to compilers. If an assembler is used, it could prevent the assembly of instructions into protected memory, but it could not detect operand addresses which would cause violations. Even if all absolute operand addresses were screened, the effective address resulting from indexing or indirect addressing could be quite different. In addition, operand addresses themselves can be modified during program execution. In view of the foregoing, it is virtually impossible for an assembler to detect beforehand when a memory violation will occur.

Quantum Timer

The quantum timer could be implemented with an elaborate real-time clock subroutine, but the highest frequency clock pulses used anywhere in the system would be required to initiate the subroutine. Since this subroutine would have to fulfill all the timing needs of the entire system, it would have a significantly long decision time. This high interrupt frequency and long decision time in combination would result in excessive overhead.

Non-interruptible Sequences

Non-interruptible sequence prevention by software is subject to restrictions similar to those noted for the above modifications. Again, a compiler can control the situation whereas an assembler cannot. It is also impossible to detect this condition in a program prior to execution. The only significant difference in this case is that such sequences are not nearly as likely to occur as privileged instructions and memory violations.

SECTION V

CONCLUSION

Interrupts have been shown to be indispensable in time-sharing computer systems. The majority of jobs performed in a time-sharing environment require a high degree of interaction with the outside world. In addition, the housekeeping involved with program switching and detection of illegal operations can both benefit from efficient interrupt systems. The demands for fast response time and low overhead dictate that time-sharing interrupt systems be as efficient as possible.

The requirements for efficiency in the proposed DCL time-sharing scheme will be much less severe than in other systems because of the small number of users. While similar operations could be performed subject to the restrictions noted in the previous section, the implementation of the proposed modifications would permit not only higher efficiency, but a vehicle for the study and development of improved time-sharing software and hardware concepts.

BIBLIOGRAPHY

1. Grabbe, Eugen M., Ramo, Simon, and Wooldridge, Dean E. Computers and Data Processing. Vol. 2 of Handbook of Automation, Computation and Control. 3 vols. New York: John Wiley & Sons, Inc, 1959.

2. Frederickson, A. A., Jr., Bailey, R. B., and Saint-Paul, A. "Hybrid Simulation of a Lifting Re-entry Vehicle," Proceedings Fall Joint Computer Conference, 1964, pp. 717-733.

3. SDS SIGMA 7 Computer Reference Manual. Publication No. 90 09 50 B, Scientific Data Systems, Inc., May 1966.

4. Naval Postgraduate School, Monterey, California, Requisition Number N62271-67-C-0013, October 13, 1966.

5. SDS 930 Computer Reference Manual. Publication No. 90 00 64D, Scientific Data Systems, Inc., February 1966.

6. Technical Manual, Computer Model 930. Publication No. 90 00 66C, Scientific Data Systems, Inc., February 1966.

7. Models 93280/90 Interrupt Control System Technical Manual. Publication No. 90 06 67A, Scientific Data Systems, Inc., July 1965.

8. SDS 940 Computer Reference Manual. Publication No. 90 06 40A, Scientific Data Systems, Inc., August 1966

9. SDS T SERIES Integrated Circuit Logic Modules, Description and Specifications. Publication No. 64 51 03R, Scientific Data Systems, Inc., 1966.

10. SDS 930 Computer Logic Equations. Publication No. 90 06 36B, Scientific Data Systems, Inc., May 1965.

11. Equations, Logic-940 Computer. Drawing No. 126185A, Scientific Data Systems, Inc., September 1966.

12. Introduction to Data Processing Maintenance Training. Publication No. 90 09 09A, Scientific Data Systems, Inc., March 1966.

13. 925/930/9300 Computers Module Reference Data. Publication No. 90 06 23C, Scientific Data Systems, Inc., February 1966.

APPENDIX I

SDS 930 OPERATION CODES

Mnemonic	Instruction Code	Name
ABC	0 46 00005	COPY A INTO B, CLEAR A
ADC	57	ADD WITH CARRY
ADD	55	ADD M TO A
ADM	63	ADD A TO M
AIR	0 02 20020	ARM INTERRUPTS
ALC 0	0 02 50000	ALERT CHANNEL W
ASC 0	0 02 12000	ALERT TO STORE ADDRESS IN CHANNEL W
BAC	0 46 00012	COPY B INTO A, CLEAR B
BETW	0 40 20010	W BUFFER ERROR TEST
BETY	0 40 20020	Y BUFFER ERROR TEST
BPT1	0 40 20400	BREAKPOINT NO. 1 TEST
BPT2	0 40 20200	BREAKPOINT NO. 2 TEST
BPT3	0 40 20100	BREAKPOINT NO. 3 TEST
BPT4	0 40 20040	BREAKPOINT NO. 4 TEST
BRM	43	MARK PLACE AND BRANCH
BRR	51	RETURN BRANCH
BRTW	0 40 21000	W BUFFER READY TEST
BRTY	0 40 22000	Y BUFFER READY TEST
BRU	01	BRANCH UNCONDITIONALLY
BRX	41	INCREMENT INDEX AND BRANCH
BTT 0, n	0 40 1201n	BEGINNING OF TAPE TEST
CAB	0 46 00004	COPY A INTO B
CAT 0	0 40 14000	CHANNEL W ACTIVE TEST; SKIP IF CHANNEL INACTIVE
CAX	0 46 00400	COPY A INTO INDEX
CBA	0 46 00010	COPY B INTO A
CBX	0 46 00020	COPY B INTO INDEX
CET 0	0 40 11000	CHANNEL W ERROR TEST; SKIP IF NO ERROR
CFT 0, 1	0 40 11006	CARD READER END-OF-FILE TEST
CIT 0	0 40 10400	CHANNEL W INTER-RECORD TEST
CLA	0 46 00001	CLEAR A
CLB	0 46 00002	CLEAR B
CLR	0 46 00003	CLEAR AB
CLX	2 46 00000	CLEAR INDEX REGISTER X
CNA	0 46 01000	COPY NEGATIVE INTO A
CPT 0, 1	0 40 14046	CARD PUNCH READY TEST
CRT 0, 1	0 40 12006	CARD READER READY TEST

Mnemonic	Instruction Code	Name
CXA	0 46 00200	COPY INDEX INTO A
CXB	0 46 00040	COPY INDEX INTO B
CZT 0	0 40 12000	CHANNEL W ZERO COUNT TEST; SKIP IF COUNT EQUALS ZERO
DIR	0 02 20004	DISABLE INTERRUPT SYSTEM
DIV	65	DIVIDE
DSC 0	0 02 00000	DISCONNECT CHANNEL W
DT2 0, n	0 40 1621n	DENSITY TEST, 200 BPI
DT5 0, n	0 40 1661n	DENSITY TEST, 556 BPI
DT8 0, n	0 40 1721n	DENSITY TEST, 800 BPI
EAX	77	COPY EFFECTIVE ADDRESS INTO INDEX REGISTER
EFT n, 4	0 02 0367n	ERASE TAPE FORWARD
EIR	0 02 20002	ENABLE INTERRUPT SYSTEM
EOD	06	ENERGIZE OUTPUT TO DIRECT ACCESS CHANNEL
EOM	02	ENERGIZE OUTPUT M
EOR	17	EXCLUSIVE OR
EPT 0, 1	0 40 14060	END OF PAGE TEST
ERT 0, n, 4	0 02 0767n	ERASE TAPE IN REVERSE
ETR	14	EXTRACT
ETT 0, n	0 40 1101n	END OF TAPE TEST
EXU	23	EXECUTE
FCT 0, 1	0 40 14006	FIRST COLUMN TEST
FPT 0 n	0 40 1401n	FILE PROTECT TEST
HLT	00	HALT
IDT	0 40 20002	INTERRUPT DISABLED TEST
IET	0 40 20004	INTERRUPT ENABLED TEST
IORD		I/O OF A RECORD AND DISCONNECT
IORP		I/O OF A RECORD AND PROCEED
IOSD		I/O UNTIL SIGNAL THEN DISCONNECT
IOSP		I/O UNTIL SIGNAL THEN PROCEED
LCY	0 67 20XXX	LEFT CYCLE AB
LDA	76	LOAD A
LDB	75	LOAD B
LDE	0 46 00140	LOAD EXPONENT
LDX	71	LOAD INDEX
LRSH	0 66 24XXX	LOGICAL RIGHT SHIFT AB
LSH	0 67 00XXX	LEFT SHIFT AB
MIN	61	MEMORY INCREMENT
MIW	12	M INTO W BUFFER WHEN EMPTY
MIY	10	M INTO Y BUFFER WHEN EMPTY
MRG	16	MERGE

Mnemonic	Instruction Code	Name
MUL	64	MULTIPLY
NOD	0 67 10XXX	NORMALIZE AND DECREMENT INDEX
NOP	20	NO OPERATION
OVT	0 40 20001	OVERFLOW INDICATOR TEST AND RESET
PBT 0, 1	0 20 12046	PUNCH BUFFER TEST
PCB 0, 1, 4	0 02 03646	PUNCH CARD BINARY
PCD 0, 1, 4	0 02 02646	PUNCH CARD DECIMAL (HOLLERITH)
PFT 0, 1	0 40 11060	PRINTER FAULT TEST
PIN	33	PARALLEL INPUT
PLP 0, 1, 4	0 02 02660	PRINT LINE PRINTER
POL 0, 1	0 02 10260	PRINTER OFF LINE
POT	13	PARALLEL OUTPUT
PPT 0, 1, 4	0 02 02644	PUNCH PAPER TAPE WITH NO LEADER
PRT 0, 1	0 40 12060	PRINTER READY TEST
PSC 0, 1, N	0 02 1N460	PRINTER SKIP TO CHANNEL N
PSP 0, 1, N	0 02 1N660	PRINTER SPACE N LINES
PTL 0, 1, 4	0 02 00644	PUNCH PAPER TAPE WITH LEADER
RCB 0, 1, 4	0 02 03606	READ CARD BINARY
RCD 0, 1, 4	0 02 02606	READ CARD DECIMAL (HOLLERITH)
RCY	0 66 20XXX	RIGHT CYCLE AB
REO	0 02 20010	RECORD EXPONENT OVERFLOW
REW 0, n	0 02 1401n	REWIND
RKB 0, 1, 4	0 02 02601	READ KEYBOARD
ROV	0 02 20001	RESET OVERFLOW
RPT 0, 1, 4	0 02 02604	READ PAPER TAPE
RSH	0 66 00XXX	RIGHT SHIFT AB
RTB 0, n, 4	0 02 0361n	READ TAPE IN BINARY
RTD 0, n, 4	0 02 0261n	READ TAPE IN DECIMAL (BCD)
RTS 0	0 02 14000	CONVERT READ TO SCAN
SFB 0, n, 4	0 02 0363n	SCAN FORWARD IN BINARY
SFD 0, n, 4	0 02 0263n	SCAN FORWARD IN DECIMAL (BCD)
SKA	72	SKIP IF M AND A DO NOT COMPARE ONES
SKB	52	SKIP IF M AND B DO NOT COMPARE ONES
SKD	74	DIFFERENCE EXPONENTS AND SKIP
SKE	50	SKIP IF A EQUALS M
SKG	73	SKIP IF A GREATER THAN M
SKM	70	SKIP IF A=M ON B MASK
SKN	53	SKIP IF M NEGATIVE
SKR	60	REDUCE M, SKIP IF NEGATIVE

Mnemonic	Instruction Code	Name
SKS	40	SKIP IF SIGNAL NOT SET
SRB 0, n, 4	0 02 0763n	SCAN REVERSE IN BINARY
SRC 0, 1	0 02 12006	SKIP REMAINDER OF CARD
SRD 0, n, 4	0 02 0663n	SCAN REVERSE IN DECIMAL (BCD)
SRR 0	0 02 13610	SKIP REMAINDER OF RECORD
STA	35	STORE A
STB	36	STORE B
STE	0 46 00122	STORE EXPONENT
STX	37	STORE INDEX
SUB	54	SUBTRACT
SUC	56	SUBTRACT WITH CARRY
TFT 0	0 40 13610	TAPE END-OF-FILE TEST
TGT 0, n	0 40 1261n	TAPE GAP TEST, CHANNEL W
TOP 0	0 02 14000	TERMINATE OUTPUT OF CHANNEL W
TRT 0, n	0 40 1041n	TAPE READY TEST
TYP 0, 1, 4	0 02 02641	WRITE TYPEWRITER
WIM	32	W BUFFER INTO M WHEN FULL
WTB 0, n, 4	0 02 0365n	WRITE TAPE IN BINARY
WTD 0, n, 4	0 02 0265n	WRITE TAPE IN DECIMAL (BCD)
XAB	0 46 00014	EXCHANGE A AND B
XEE	0 46 00160	EXCHANGE EXPONENTS
XMA	62	EXCHANGE M AND A
XXA	0 46 00600	EXCHANGE INDEX AND A
XXB	0 46 00060	EXCHANGE INDEX AND B
YIM	30	Y BUFFER INTO M WHEN FULL

APPENDIX II

SDS 930 LOGIC TERMINOLOGY

The SDS 930 computer logic was designed to fulfill two major objectives: economy and reliability. Economical design was achieved by using a large variety of logic modules, each tailored more or less to a particular need. Reliability was achieved by using carefully selected components, including all silicon transistors.

This appendix will describe the characteristics of the SDS 930 computer logic circuits, logic equations and timing diagrams.

I. LOGIC CIRCUITRY

Logic Levels

Logic signals are represented in the SDS 930 by DC logic levels as follows:

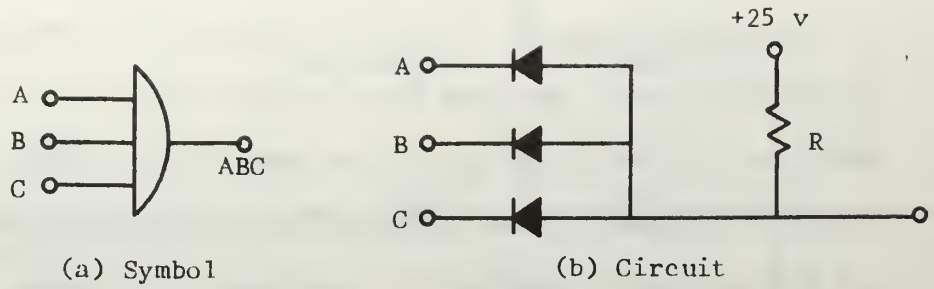
0 volts = 0 = False

+4 volts = 1 = True

Most of the SDS 930 series peripheral equipment uses +8 volts for the True signal, but the necessary conversions are made when this equipment communicates with the computer.

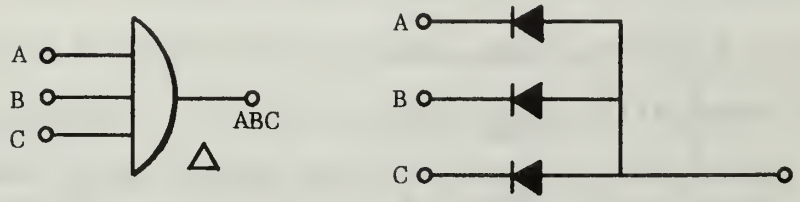
Logic Gate Circuitry

AND gate. The AND gate must have true signals on all inputs in order for the output to be true. The standard symbol and the circuit used to implement the AND gate are shown in Figure 3. If any one of the diodes has a false signal applied to its cathode, that diode will conduct current from the load resistor, R, and cause the output to drop to 0 volts. If all diodes have true inputs, the IR drop through R will be 21 volts. The output will then be +4 volts, or true.



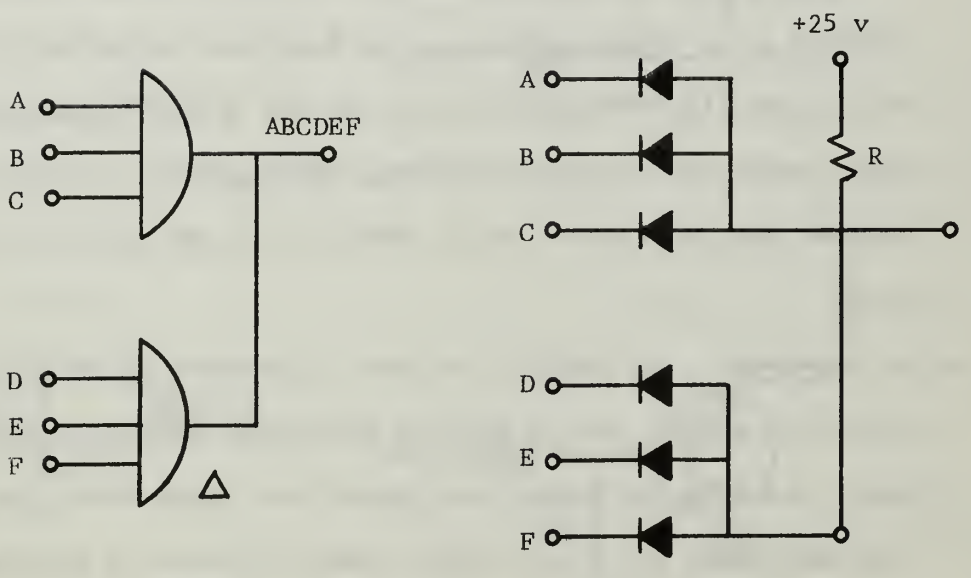
(a) Symbol (b) Circuit

Figure 3. AND Gate.



(a) Symbol (b) Circuit

Figure 4. Expander AND Gate.



(a) Symbol (b) Circuit

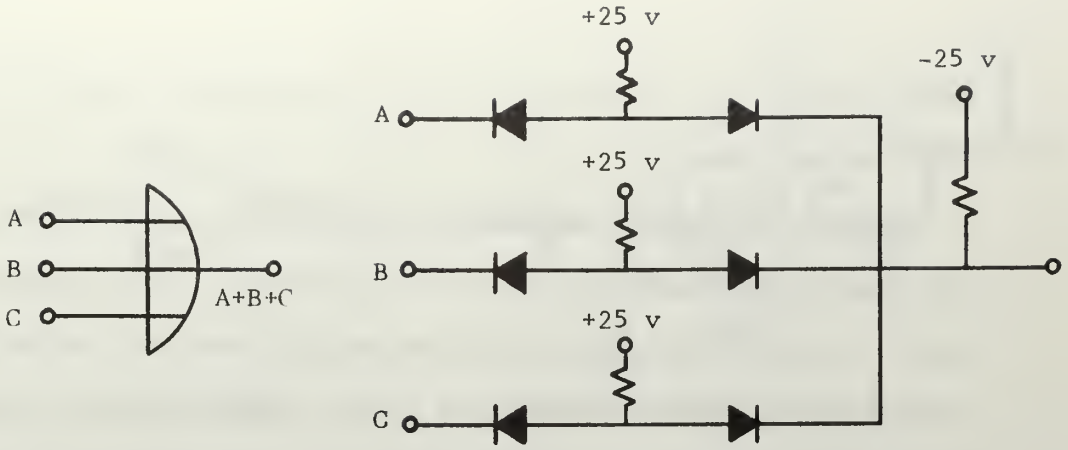
Figure 5. Expanded AND Gate.

Expander AND gate. The expander AND gate is identical to the AND gate except that it does not have the load resistor. It is used to add inputs to a standard AND gate. The symbol and circuit are shown in Figure 4. Figure 5 shows the method of expanding an AND gate. A standard AND gate may be expanded to a maximum of 30 inputs.

OR gate. The OR gate will have a true output if any one or more of its inputs are true. This function is illustrated in Figure 6. Each of the inputs is fed to a single input AND gate, which in turn is connected to an output diode. If one of the inputs is true, the associated output diode will have +4 volts at its anode. Since the forward resistance of the diode is negligible, the voltage at the bottom of the output load resistor is +4 volts, and the output is therefore true. All input connections must be connected to desired signal sources or grounded. If an input is left open, the output will always be true.

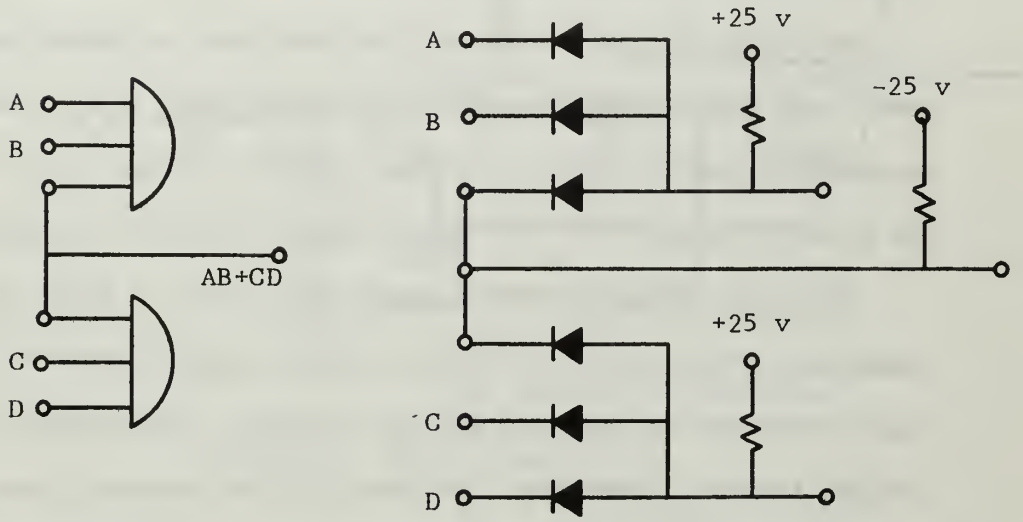
AND/OR gate. When the outputs from two or more AND gates are to be combined at an OR gate, this can be done most economically by using an extra input diode to each AND gate as the OR gate output diodes. The logic symbol and circuitry for this configuration are shown in Figure 7. Expander AND gates may also be added to the AND portions of the AND/OR gate.

Inverter. An inverter is used to generate the complement or inverse of a signal, and to provide additional driving capability to signals. The inverter symbol and circuit are shown in Figure 8. The circle superimposed on any logic symbol always indicates a complement operation. Since the Zener diode is always back-biased by R_b, the base of the transistor will be 3.3 volts below the input. When the input is 0 volts, the transistor is cut off, and the output is +4 volts, or true. When the



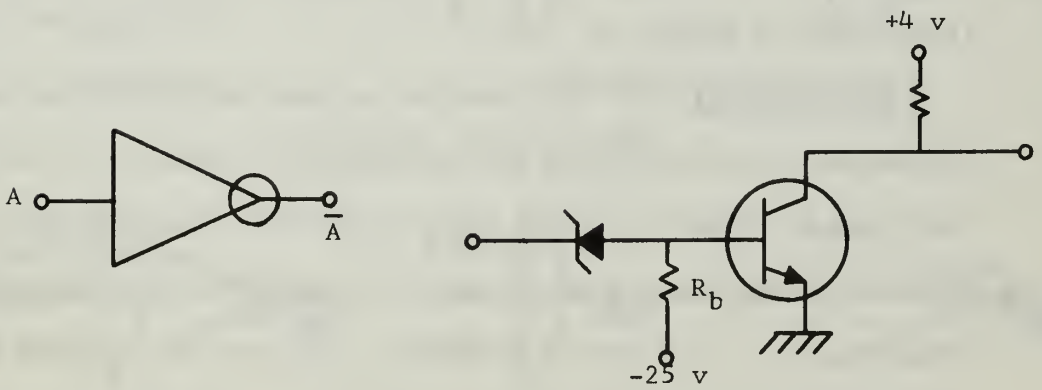
(a) Symbol (b) Circuit

Figure 6. OR Gate.



(a) Symbol (b) Circuit

Figure 7. AND/OR Gate.



(a) Symbol (b) Circuit

Figure 8. Inverter.

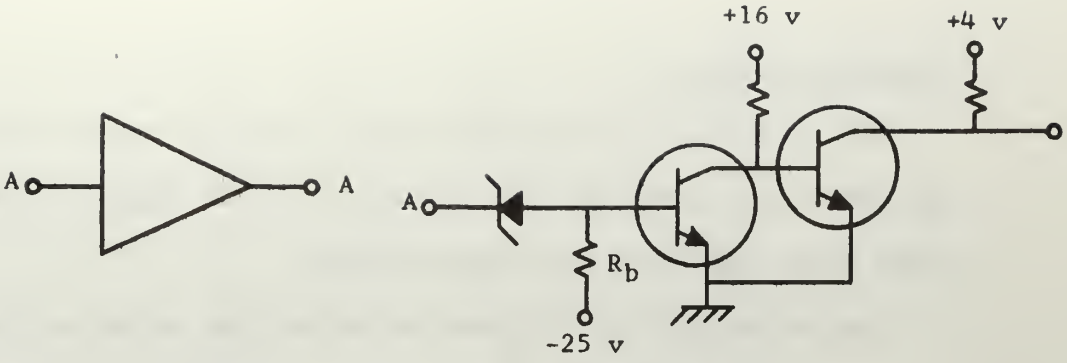
input is true, the transistor is in full conduction, and the output is false (about .25 volt).

Buffer amplifier. The buffer amplifier is used to increase the fan-out capacity of diode logic gates. It consists of two inverters in cascade. In some cases the Zener diode and bias resistor of the second inverter are omitted. The circuit symbol and both circuit configurations are shown in Figure 9. The operation of each inverter is the same as described in the previous paragraph.

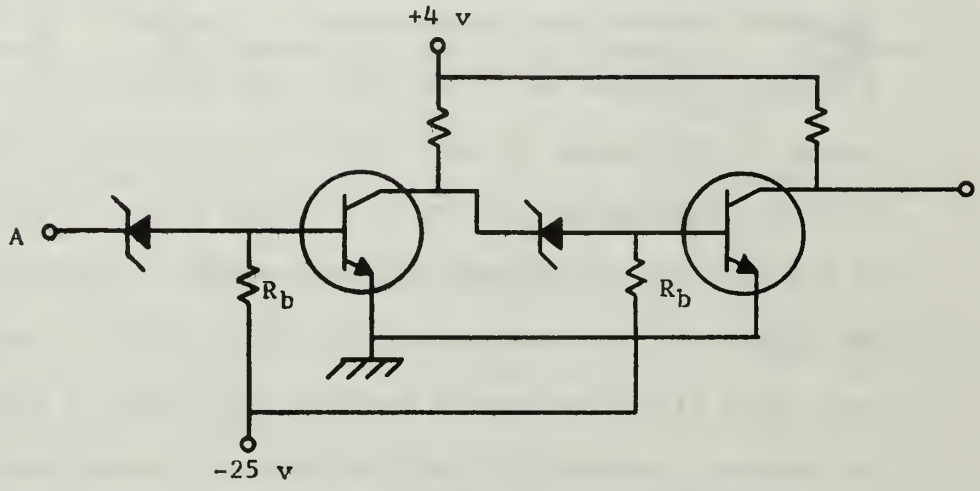
Buffered AND gate. The buffered AND gate, or BAND, performs the same logic function as the AND gate, except that it provides for increased fan-out capability. It is a series combination of an AND gate and a buffer amplifier. The logic symbol is shown in Figure 10.

Negative AND gate. The negative AND gate, or NAND, produces the complement of the standard AND gate as an output, since it is a series combination of the AND gate and the inverter. The output also has increased fan-out capability. The NAND is the standard logic circuit used in the SDS 930. If there is a choice between a NAND and some other circuit which will do the same job, the NAND is chosen. The NAND symbol is illustrated in Figure 11.

Line inverter. A line inverter is used on each end of a coaxial transmission line when used for transmitting signals. It is similar to the standard inverter, except that it performs the additional tasks of impedance matching, and logic level modification where required. The inverter at the sending end is called a line driver, and the one at the receiving end is called a line receiver. The symbols for these are shown in Figure 12.

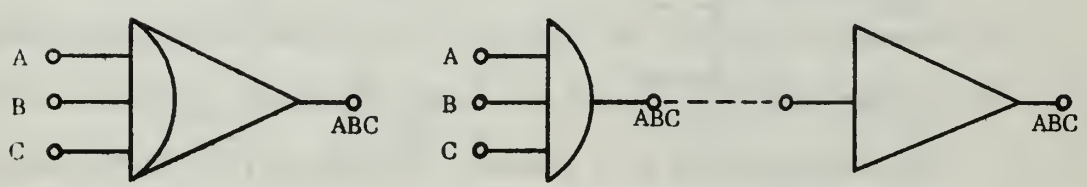


(a) Symbol (b) First Configuration



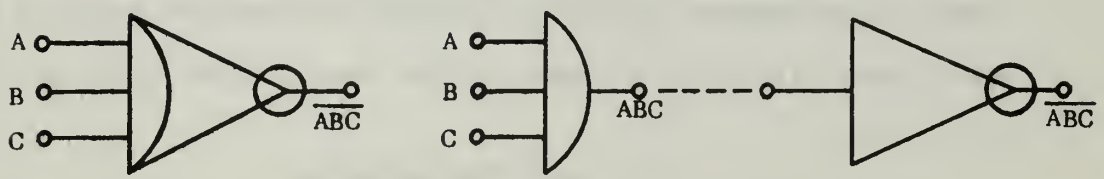
(c) Second Configuration

Figure 9. Buffer Amplifier



(a) Symbol (b) Equivalent Circuit

Figure 10. Buffered AND.



(a) Symbol (b) Equivalent Circuit

Figure 11. Negative AND.

Flip-flop Operation

The flip-flop is the primary logic signal storage device in the SDS 930. Many configurations are used, but the logical operation of three basic types will be described here.

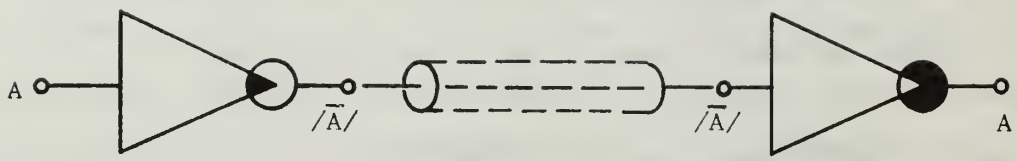
The standard flip-flop used in the SDS 930 is the AC flip-flop. This circuit is a standard transistorized bistable multivibrator, with the added feature that its operation is controlled by timing pulses. If a signal is present at the set input and the flip-flop is in the reset state, or vice versa, it will switch to the opposite state in coincidence with the trailing edge of the next timing pulse. This provision makes it possible for a flip-flop to have separate, independent, input and output signals at the same time with no possibility of confusion between the two. This flip-flop usually contains a DC input to either the set or reset states. Grounding of this DC input will cause the flip-flop to switch to the indicated state immediately without regard to timing pulses.

The NAND flip-flop is similar to the one previously described, with the exception that the set and reset inputs are fed through NAND gates. The circuit symbol of the AC and NAND flip-flops are shown in Figure 13.

The DC flip-flop is a standard bistable multivibrator without AC clocking. The set input is normally a series of three AND gates, with two inputs each, combined in an OR gate. DC flip-flops are used as the stages of the memory register to provide the shortest possible memory cycle time. Its circuit symbol is the same as that of the AC flip-flop. [12]

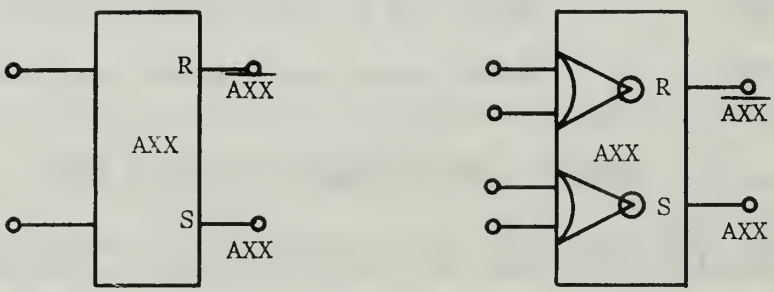
II. LOGIC DOCUMENTATION

The construction and operation of the computer are fully described by (1) the logic equations, (2) the logic diagrams, (3) the timing diagrams, and (4) the module reference data. Each will be described briefly, followed by an example function illustrating the use of these documents.



(a) Line Driver (b) Line Receiver

Figure 12. Line Inverters.



(a) AC Flip-flop (b) NAND Flip-flop

Figure 13. Flip-flops

Logic Equation

Logic equations are used to define the origin of each signal generated within the computer. The left-hand side of the equation is a single term which names the signal generated by the combination of terms on the right-hand side. Some signals involving registers have names which are descriptive of the function of the signal. These are illustrated in Table 4 using a fictitious D register.

Term	Meaning
sD12	Sets flip-flop D12, the 12th stage of the D register
rD12	Resets flip-flop D12
Dc	Clears (resets) the D register
Dk	Complements the D register
Dxe	Enables the transfer of the contents of the E register to the D register
Dr3	Shifts the D register right 3 bits
Dℓ2	Shifts the D register left two bits

Table 4. Special Logic Equation Terms

The right-hand side of a logic equation represents the logical combination of signals which produces the generated signal. The AND function is represented by the multiplication of terms, the OR function is represented by the addition of terms, and the NOT function is represented by a bar over the applicable terms.

Logic Diagram

The CPU consists of circuits mounted on logic modules. There are five rows of 64 modules each. The rows are lettered B, C, D, E, and F. A logic element numbered 32C is on the 32nd module of the C, or second, row.

The logic diagrams show the circuitry, in logic symbol form, which is used to implement each logic equation. These diagrams also show the location of each circuit in the computer for maintenance purposes. A portion of a typical logic diagram is shown in Figure 14. This circuit consists of one NAND gate into the set and one into the reset sides of the NAND flip-flop Mgz.

The numbers adjacent to the small circles in Figure 14 indicate pin numbers on the logic modules. The set output of Mgz has a separate load resistor, which is mounted on logic module 44E and connected to pin number 40 of that module. The flip-flop is located on module number 40C, and the NAND gates on 54C.

The logic equations for Mgz are

$$sMgz = Zrq\ T3\ \overline{St}$$

$$rMgz = T4\ .$$

The correlation between the equations and the circuit is readily apparent.

Timing Diagram

The timing diagrams indicate the sequence of operations initiated by each computer instruction. A sample timing diagram is shown in Figure 15. The space between two horizontal lines represents one timing pulse. Each signal initiated by a timing pulse is shown to the right of a vertical line in the space for that pulse. The timing pulse for each space and the particular phase in which it occurs are labeled at the left-hand edge of the diagram. If the signal is present for more than one timing pulse, this is indicated by a vertical line through all of the applicable timing pulse spaces.

For example, in Figure 15, the signal sEax is true during timing pulse T8 of phase zero only. The signals C→Add, Cr3, and Add→C are all true during timing pulses T7 through T0 of phase zero.

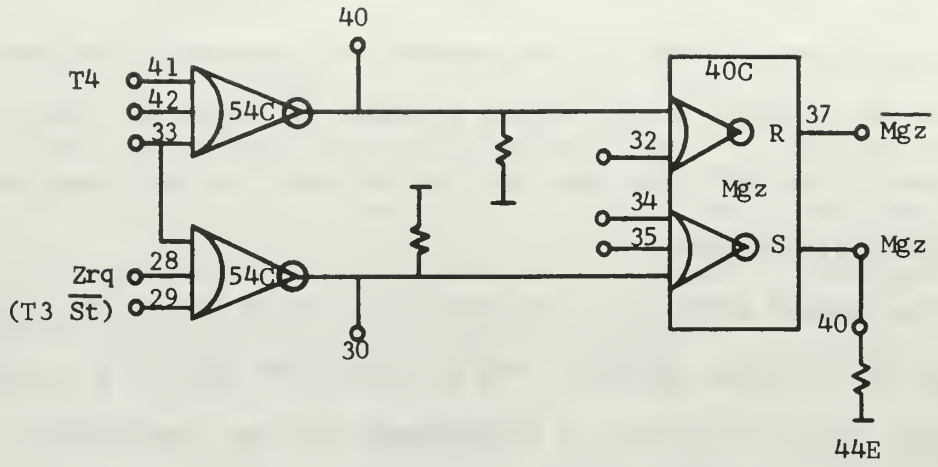


Figure 14. Typical Logic Diagram

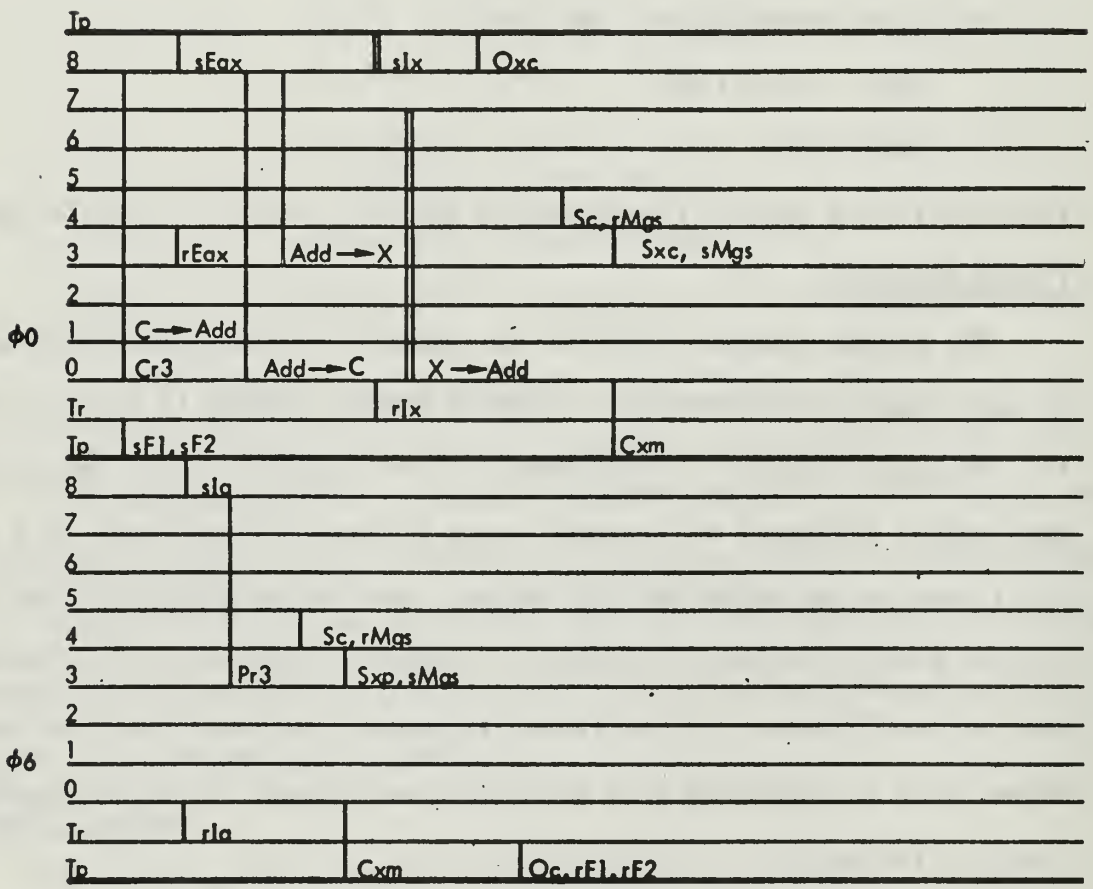


Figure 15. Timing Diagram, EAX Instruction

The double vertical line indicates a conditional action. In Figure 15, if Ix is set at T8, then X→Add is true from T6 through T0. If Ix is not set, then X→Add will remain false.

Module Reference Data

The module reference data publication contains a complete circuit diagram for each different type of logic module used in the SDS 930 computer. [13]

Example

The use of the Ju flip-flop in a Branch Unconditional instruction, BRU, will be traced to illustrate the method of using the previously described documents. The BRU causes the next instruction to be taken from the memory location indicated by the operand address.

The timing diagram for the BRU is shown in Figure 16. It shows that the Ju flip-flop is set at T8 of Ø0. This is verified by the logic equation for setting Ju:

$$sJu = (Ø0\ T8\ \overline{Ia}\ Go)\ C2 + (Ø0\ T8\ \overline{Ia}\ Go)\ \overline{C4}\ \overline{C5}\ C8\ \overline{C9} + (Ø0\ T8\ \overline{C9})\ \overline{02}\ \overline{03}\ .$$

The second term of the equation is applicable in this case. It indicates that if indirect addressing is not being used ($\overline{Ia}$ and $\overline{C9}$), AND the computer is operating (Go), AND the proper opcode is present ($\overline{C4}$ $\overline{C5}$ C8), then Ju is set at T8 of Ø0.

The implementation of the sJu logic equation into circuitry is shown in Figure 17. The output of the NAND gate at pin 28 of module 47B is $\overline{(Ø0\ T8\ \overline{Ia}\ Go)\ C2}$. The direct combination of this NAND gate and the other two NAND gates with load resistor at pin 51 of module C9 constitutes an AND function. The input to the NAND gate at pin 6 of module 46B is

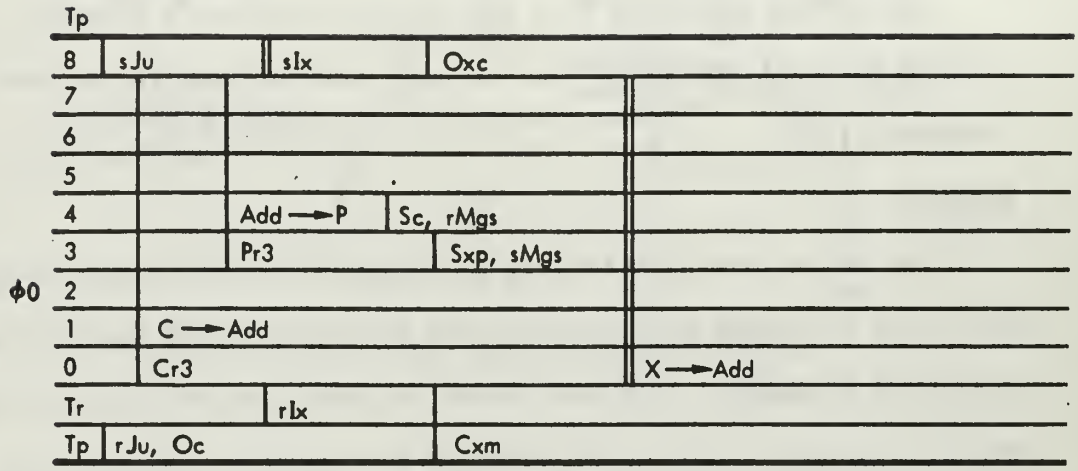


Figure 16. Timing Diagram, BRU Instruction.

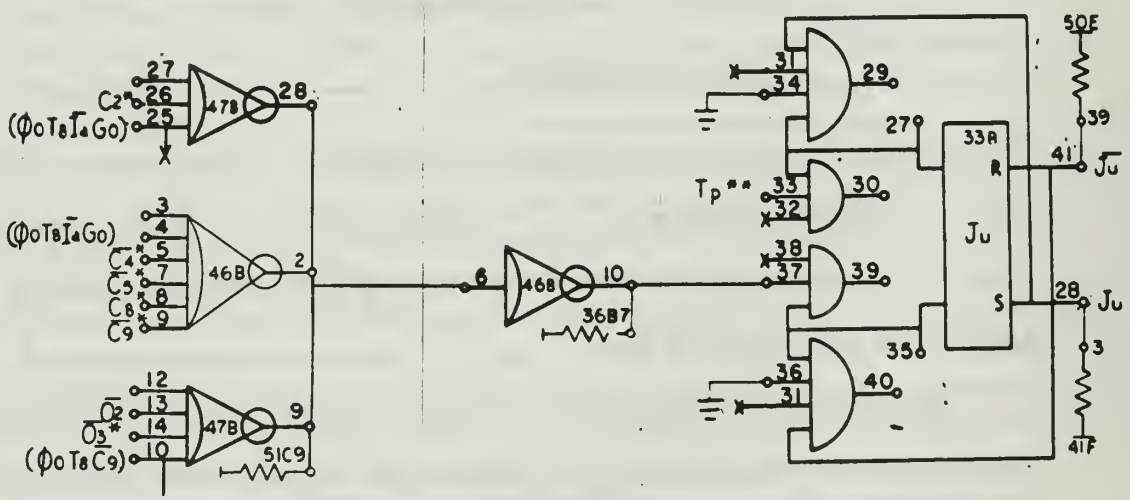


Figure 17. Ju Flip-flop Logic Circuit.

therefore:

$$\overline{(\emptyset 0\ T8\ \overline{Ia}\ Go\ C2)}\ \overline{(\emptyset 0\ T8\ \overline{Ia}\ Go\ \overline{C4}\ \overline{C5}\ C8\ \overline{C9})}\ \overline{(\emptyset 0\ T8\ \overline{C9}\ \overline{02}\ \overline{03})}\ .$$

By DeMorgan's theorem, this is equivalent to:

$$\overline{(\emptyset 0\ T8\ \overline{Ia}\ Go\ C2) + (\emptyset 0\ T8\ \overline{Ia}\ Go\ \overline{C4}\ \overline{C5}\ C8\ \overline{C9}) + (\emptyset 0\ T8\ \overline{C9}\ \overline{02}\ \overline{03})}\ .$$

The output of the NAND gate, which is the inverse of the above function, is the necessary signal to set flip-flop Ju and is the input to pin 37 of module 33B.

The actual circuit used can be determined as follows. The Ju flip-flop is located on the module at location 33B. The diagram on page 5 of the logic diagram publication shows that this location contains an FB52 logic module. The complete circuit diagram for this module is shown in the module reference data publication.

APPENDIX III

SUMMARY OF PROPOSED LOGIC MODIFICATIONS

$$sAi = \left\{\left[Ir(En + \overline{En}) + Is\right] \emptyset 0\ T0\ \overline{Int}\ Mo \left[Ia\ (01\ \overline{02}\ \overline{03}\ 04)\ (\overline{01}\ 02\ \overline{03}) + \overline{01}\ 02\ \overline{03}\ \overline{04}\ 05\ 06\right]\right\}$$

$$rAi = T3$$

$$Cr3 = Tsr\ Q1 + \overline{F1}\ \overline{F2}\ \overline{Ts}\ Q1 + F1\ \overline{F3}\ \overline{Ts}\ Q1 + \emptyset 7\ \overline{05}\ Ts\ Q1 + Ex\ \overline{Ts}\ Q1 + St + \overline{Kcr + Kmc}\ \overline{Ts}\ Q1 + \overline{Go}\ \overline{Ix}\ Rf + \emptyset 5\ 0b\ 06\ \overline{Ts}\ Q1$$

$$sEax = (\emptyset 0\ T8\ \overline{Ia}\ Go\ C2 + \emptyset 0\ T8\ \overline{Ia}\ Go\ C3\ C4\ C5\ C6\ C7\ C8\ \overline{C9} + \emptyset 0\ T8\ \overline{C9}\ \overline{02}\ \overline{03})\ \overline{Ob}\ \overline{Pi}\ \overline{Ai}\ \overline{Piq}$$

$$sF1 = \overline{Ob}\ Tp\ (Eax + Sk + \emptyset 4) + \overline{Ob}\ \emptyset 0\ Tp\ \overline{Ia}\ (03\ 04 + 01\ \overline{04}) + \overline{Ob}\ Tp\ \overline{F1}\ \overline{F3}\ \overline{01}\ 03\ \overline{04}\ \overline{Ia}\ Rf + \emptyset 0\ T8\ \overline{Ia}\ \overline{C2}\ \overline{C5}\ \overline{C8}\ (\overline{C3} + \overline{C4}) + T8\ (Ob + Ai) + Piq + T8\ \overline{Go}$$

$$rF1 = Tp\ (End\ \overline{Sk} + Ob)$$

$$
\begin{aligned}
sF2 = \;& \overline{Ob}\ Tp\ (Eax + Sk + \emptyset 4) \\
& + \overline{Ob}\ \emptyset 0\ Tp\ \overline{Ia}\ 01\ \overline{02} \\
& + \overline{Ob}\ \emptyset 0\ Tp\ \overline{Ia}\ 03\ (01 + \overline{02}) \\
& + \overline{Ob}\ \emptyset 0\ Tp\ \overline{Ia}\ 03\ \overline{04}\ \overline{Rf} \\
& + \emptyset 1\ Tp \\
& + \emptyset 1\ 05\ Q2 \\
& + \overline{Ob}\ \emptyset 0\ Tp\ \overline{Ia}\ 04\ \overline{05}\ \overline{06}
\end{aligned}
$$

$$
\begin{aligned}
sF3 = \;& \overline{Ob}\ Tp\ (Eax + Sk + \emptyset 4) \\
& + \emptyset 0\ T8\ \overline{Ia}\ \overline{C2}\ \overline{C5}\ \overline{C8}\ (\overline{C3} + \overline{C4}) \\
& + T8\ Go \\
& + \overline{Ob}\ \emptyset 0\ Tp\ \overline{Ia}\ \overline{03}\ 04 \\
& + \emptyset 0\ \overline{Ia}\ Q4\ \overline{03}\ 04\ 05 \\
& + T8(Ob + Ai) \\
& + Piq
\end{aligned}
$$

$$
\textcircled{I24} = Mo\ Qt4
$$

$$
\begin{aligned}
sIa = \;& \emptyset 0\ T8\ \overline{C2}\ C9\ \overline{\left[\emptyset 0\ \overline{Ia}\ T8\ \overline{C2}\ \overline{C5}\ \overline{C8}\ (\overline{C3} + \overline{C4})\right]} \\
& + \left[\emptyset 0\ \overline{Ia}\ T8\ \overline{C2}\ \overline{C5}\ \overline{C8}\ (\overline{C3} + \overline{C4})\right]\ \textcircled{Kr}\ \overline{Pi}\ \overline{Ai}\ \overline{Ob}\ \overline{Piq} \\
& + T8\ \emptyset 7\ Sk\ \textcircled{Kr}\ (\overline{Ij} + Inr) \\
& + T8\ F1\ \overline{F3}\ \textcircled{Kr}\ (\overline{Ij} + Inr) \\
& + T8\ \textcircled{Kmc}
\end{aligned}
$$

$$
\begin{aligned}
rIa = \;& \overline{(P12\ P13\ P14)}\ Q2\ F1 \\
& + Tr\ F1\ (\overline{Ob}\ \overline{Pi}\ \overline{Ai}\ \overline{Piq}) \\
& + \emptyset 0\ T8\ \overline{C9}\ Ia \\
& + Ai\ Tr
\end{aligned}
$$

sInt = $\overline{(\emptyset 5\ 01\ 05\ \overline{Ts}\ Q1)}\ T4\ End\ \ Is + Ir(En + En\ )\ \overline{Ts}\ \overline{Ob}\ \overline{Pi}$

sJu = (Ø0 T8 $\overline{Ia}$ Go C2 + Ø0 T8 $\overline{Ia}$ Go $\overline{C4}$ $\overline{C5}$ C8 $\overline{C9}$
+ Ø0 T8 $\overline{C9}$ $\overline{02}$ $\overline{03}$) $\overline{Ob}$ $\overline{Pi}$ $\overline{Ai}$ $\overline{Piq}$

sMo = Eom C10 $\overline{C11}$ C18 TO
+ Usi Ib

rMo = T3 (Pi + Ob)
+ Tr Mo Int

Mxc = $\overline{Ob}$ [$\overline{Tsm}$ (Ø4 + Eax + Ju 01 05) + Tsm $\overline{R9}$ + (Kmc)]

s06 = Oxc C8 + Oc Ob Ju 01 05

r06 = Oc $\overline{Ob}$

sOb = Pme Mo End T2 $\overline{Int}$
+ Pme Mo Ø0 T2 [$\overline{(01\ \overline{02}\ \overline{03}\ 04)}\ \overline{(\overline{01}\ 02\ \overline{03})}\ \overline{(01\ 02\ 03\ 04\ 05\ 06)}$
$\overline{(Ju\ 01\ \overline{05}\ \overline{Xw1})}$]

rOb = T3

Oc = Tp End $\overline{Sk}$ + Tp $\overline{01}$ $\overline{03}$ $\overline{Ia}$ Go + Tp (Ob + Ai)

Oxc = Ø0 T8 $\overline{Ia}$ Go $\overline{C2}$ $\overline{Piq}$ $\overline{Ob}$ $\overline{Ai}$

sPO = Pr3 {Add1 (Ju $\overline{Eax}$ + $\overline{02}$ $\overline{04}$ $\overline{05}$ 06 + Ob 06) + C6 [Eax (T7 + T6)]
+ P12 $\overline{(P13\ P14\ Ia)}$ F1 (Go + (Kmc)) $\overline{\overline{(02\ 04\ 05\ 06)}}$
+ $\overline{P12}$ (P13 P14 Ia) F1 (Go + (Kmc)) $\overline{\overline{(02\ 04\ 05\ 06)}}$}

rPO = Pr3{ $\overline{\qquad\qquad}$ }

$$sP1 = Pr3\{Add2\ (Ju\ \overline{Eax} + \overline{02}\ \overline{04}\ \overline{05}\ 06 + Ob\ 06) + C7[Eax\ (T7 + T6)]$$
$$+ P13\ \overline{(P14\ Ia)}\ F1\ (Go + \textcircled{Kmc})\ \overline{(\overline{02}\ \overline{04}\ \overline{05}\ 06)}$$
$$+ \overline{P13}\ (P14\ Ia)\ F1\ (Go + \textcircled{Kmc})\ \overline{(\overline{02}\ \overline{04}\ \overline{05}\ 06)}\}$$

$$rP1 = Pr3\{\quad\}$$

$$sP2 = Pr3\{Add3\ (Ju\ \overline{Eax} + \overline{02}\ \overline{04}\ \overline{05}\ 06 + Ob\ 06)$$
$$+ C8\ [Eax\ (T7 + T6)]$$
$$+ P14\ \overline{Ia}\ F1\ (Go + \textcircled{Kmc})\ \overline{(\overline{02}\ \overline{04}\ \overline{05}\ 06)}$$
$$+ \overline{P14}\ Ia\ F1\ (Go + \textcircled{Kmc})\ \overline{(\overline{02}\ \overline{04}\ \overline{05}\ 06)}$$
$$+ Eax\ T5\}$$

$$rP2 = Pr3\{\quad\}$$

$$sPi = Piq$$

$$rPi = Pi\ T2$$

$$Pid = \overline{C3}\ \overline{C5}\ C6$$
$$\overline{C3}\ C5\ \overline{C6}$$
$$\overline{C4}\ \overline{C5}\ \overline{C6}\ \overline{C8}$$
$$\overline{C3}\ \overline{C4}\ \overline{C5}\ C7$$
$$\overline{C4}\ \overline{C5}\ \overline{C7}\ \overline{C8}$$
$$\overline{C4}\ \overline{C5}\ C6\ C8$$
$$\overline{C3}\ \overline{C4}\ C5\ \overline{C7}\ C8$$
$$\overline{C3}\ C4\ \overline{C5}\ \overline{C7}\ C8$$
$$\overline{C3}\ C4\ C5\ \overline{C7}\ \overline{C8}$$
$$\overline{C3}\ C4\ \overline{C5}\ C7\ \overline{C8}$$

$$Piq = Pid\ Mo\ Go\ \overline{C2}\ Ø0\ T8\ \overline{Ia}$$

$$
\begin{aligned}
Pme &= (Mp1)\ \overline{S1} \\
&\quad + (Mp1)(Mp2)\ \overline{S2} \\
&\quad + (Mp2)\ \overline{S1}\ \overline{S2} \\
&\quad + (Mp1)(\overline{Mp2})\ (Mp3)\ \overline{S2}\ \overline{S3} \\
&\quad + (Mp1)(Mp2)\ (Mp3)\ S2\ \overline{S3} \\
&\quad + (\overline{Mp2})(Mp3)\ \overline{S1}\ \overline{S2}\ \overline{S3} \\
&\quad + (Mp2)(Mp3)\ \overline{S1}\ S2\ \overline{S3}
\end{aligned}
$$

$$sQt3 = \overline{Qt3}\ \overline{Qt2}\ Ti$$

$$rQt3 = Qt3\ Qt4\ Ti + (St)$$

$$sQt4 = Qt3\ \overline{Qt4}\ Qt2\ Ti$$

$$rQt4 = Qt4\ Ti + (St)$$

$$
\begin{aligned}
sS9 &= Sxc\ C6 \\
&\quad + Sxp\ P6 \\
&\quad + Sxn\ N9 \\
&\quad + Sx48 \\
&\quad + T3\ (Pi + Ob)
\end{aligned}
$$

$$
\begin{aligned}
sS14 &= Sxc\ C11 \\
&\quad + Sxp\ P11 \\
&\quad + Sxn\ N14 \\
&\quad + Ø3\ 05\ \overline{06}\ \overline{Sk}\ \overline{S14} \\
&\quad + Ø3\ 05\ 06\ \overline{Sk}\ C5\ (A1\ \overline{A2} + \overline{A1}\ A2) \\
&\quad + T3\ Ob
\end{aligned}
$$

$$Ski = Sk\ Ø7\ Ij\ \overline{Inr}\ Ip8$$

Sku = Sk Ø7 Ij $\overline{\text{Inr}}$ Ip24

Sxp = T3 $\overline{\text{Int}}$ (End $\overline{\text{Pi}}$ $\overline{\text{Ob}}$ + Ju $\overline{\text{Eax}}$) Go

+ T3 (Kmc)

sUsi = Mo Int Ij Tr

rUsi = Usi Ib

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library Naval Postgraduate School Monterey, California 93940	2
3.	Director of Naval Communications Department of the Navy Washington, D.C. 20350	1
4.	Prof. M. L. Cotton Dept. of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
5.	LCDR James White Egerton, USN USS GARCIA (DE-1040) Fleet Post Office New York, N.Y. 09501	1

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE
The Use of Interrupts in a Time-Sharing Computer System

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Thesis, M.S., June 1967

5. AUTHOR(S) *(Last name, first name, initial)*
EGERTON, James White

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1967	89	13

8a. CONTRACT OR GRANT NO.

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES
This document has been approved for public release and sale; its distribution is unlimited.
~~This document is subject to special export controls and each transmittal to foreign nationals may be made only with prior approval of the Naval Postgraduate School.~~

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Director of Naval Communications
Department of the Navy
Washington, D.C. 20350

13. ABSTRACT

Digital computer interrupts are becoming more important as these machines increase in the interaction with their environment. Different methods of interrupt implementation are described. They are then analyzed in the areas of response time, overhead, and saturation. Examples of the use of interrupts in different computational environments are given. Five modifications to a general purpose computer system are proposed. These modifications, each of which used interrupts, enable the system to be more easily used in a limited time-sharing mode. The results of these modifications are compared to those of the software that would be required to accomplish the same objectives.

DD FORM 1 JAN 64 1473

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Time-sharing						
Interrupt						